MASTER EN SCIENCES INFORMATIQUES

Étude des possibilités de UNIX appliquées à la gestion

Bodart, Jean-Pierre; Tixhon, Jean-Paul

Award date:
1981

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Mémoire présenté par

Jean-Pierre Bodart
&
Jean-Paul Tixhon

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

ETUDE DES POSSIBILITES DE UNIX

APPLIQUEES A LA GESTION

Mémoire présenté par

Jean-Pierre Bodart
&
Jean-Paul Tixhon

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

Année académique 1980-1981

TABLE DES MATIERES

REMERCIEMENTS.

Nous tenons tout d'abord à remercier Monsieur Ph. van Bastelaer pour avoir bien voulu diriger ce mémoire.

Nous remercions également Monsieur E. Milgrom dont les conseils se sont révélés judicieux pour la réalisation de notre travail.

Nous exprimons notre gratitude à Messieurs F. Bodart, J-L. Hainaut et à A. van Lamsweerde pour avoir bien voulu participer à quelques réunions de travail.

Nous remercions tout particulièrement Madame M-F. Declerfays et Messieurs S. Allemon et H. Broze pour la disponibilité qu'ils nous ont manifestée lors de notre stage.

Enfin, notre reconnaissance va à Messieurs J-P. Adans et R. Verhaeghe pour l'aide qu'ils nous ont apportée lors de notre travail à l'Institut d'informatique.

INTRODUCTION.

Introduction.

Dans le cadre de nos études de licence et maitrise en informatique, nous avons abordé l'analyse fonctionnelle et l'analyse organique d'une application de gestion (dénommée ci-dessous application Petitpas).Il est apparu intéressant de réaliser au moins partiellement une implémentation de cette application.

Cette implémentation était à effectuer sous le système d'exploitation UNIX(1),dont la conception n'a pas été originellement orientée vers les applications de gestion,et dans le langage PASCAL.Ce travail devait nous permettre de déterminer et de réaliser les outils informatiques nécessaires à l'implémentation d'une application de gestion type sous ce système d'exploitation

Le but principal de ce travail n'était donc pas tellement la réalisation complète et rigoureuse de Petitpas,mais bien la recherche des primitives à ajouter à UNIX et leur réalisation.La preuve du caractère satisfaisant de ces primitives est montrée par l'implantation d'un sous-ensemble de Petitpas. Ce sous-ensemble a été choisi de telle façon,qu'il fasse apparaître la totalité des problèmes qu'il nous semblait trouver dans l'entièreté du cas.

Le premier chapitre du mémoire donne le contexte général du problème que nous avons traité.

La connaissance de l'application Petitpas et de UNIX étant indispensable pour comprendre la suite,nous consacrons le deuxième et le troisième chapitre à leur présentation. C'est dans ce deuxième chapitre que l'on donne brièvement les caractéristiques informatiques essentielles de l'application.

Viennent ensuite,les chapitres décrivant l'analyse et la réalisation du cas Petitpas.

Cette analyse suit,dans la limite du possible,la démarche enseignée à l'Institut d'informatique.Ce choix méthodologique fut pris pour nous familiariser avec la méthode,mais également parce qu'une partie de cette analyse était déjà réalisée.Il n'était pas possible,dans le cadre de notre mémoire, de développer de A à Z,un projet et de trouver une solution à tous les problèmes rencontrés.Certains de ces chapitres seront donc une reprise de ce qui existait déjà;mais nous indiquerons ce qui a été traité et modifié dans le cadre de notre travail.

---

(1) UNIX est une marque déposée par les Bell Laboratories.

Le quatrième chapitre présente l'analyse fonctionnelle de notre projet. Nous rappelons d'abord, le but de l'analyse fonctionnelle et la méthode choisie pour la réaliser. Nous exposons ensuite comment nous avons déterminé le sous-ensemble des spécifications fontionnelles pour lequel une implémentation fut effectuée.

Dans le cinquième chapitre, et de nouveau après la description du but et de la méthode choisie, nous présentons de manière détaillée les résultats de l'analyse organique d'une part en ce qui concerne la structure des données et d'autre part en ce qui concerne l'architecture des programmes.

Nous expliquons au chapitre six, la façon dont nous avons décidé d'implémenter le sous-ensemble de Petitpas, tant au point de vue de la structure fine des fichiers que de celle des programmmes.

De cette étude, nous déduisons les principales primitives nécessaires et, par comparaison avec les possibilités du système d'exploitation (UNIX) et du langage (PASCAL), la liste des primitives supplémentaires à écrire.

Le chapitre suivant présente de manière appronfondie la réalisation de ces primitives nouvelles de UNIX. Nous nous sommes particulièrement attachés à créer des outils permettant l'accès direct et séquentiel aux fichiers et la communication entre processus.

Le chapitre huit montre quelques aspects de conduite du système informatique relatif à Petitpas : démarrage, arrêt, accès et maintenance du système.

Ce mémoire se termine par un chapitre contenant des suggestions sur ce qu'il nous semble encore indispensable d'apporter au système UNIX pour répondre à nos préoccupations, et sur quelques améliorations intéressantes de nos primitives.

CHAPITRE 1 : SITUATION DU PROBLEME.

1. Situation du problème.

Dans ce chapitre,nous présentons ,en première partie,ce dont nous disposions et ce que nous avons réalisé,et en seconde partie nous exposons le déroulement du travail.

Le sujet traité.

Au départ,nous disposions de l'analyse fonctionnelle de Petitpas,de l'analyse organique de la base de données ainsi qu'un début d'analyse organique de l'architecture des programmes.

Nous avons choisi le sous-ensemble de Petitpas nous permettant de mettre en évidence tous les problèmes que la réalisation d'une telle application fait apparaître.Pour ce sous-ensemble,nous avons affiné l'analyse organique tant au niveau de l'architecture des programmes qu'au niveau de la spécification et de la conception des modules de programme.

A partir de cette analyse,nous avons pu déterminer la liste des outils qui nous semblaient indispensables pour réaliser l'implémentation de l'architecture choisie. Nous devions disposer d'outils d'accès et de manipulation de fichier, d'outils de communication de messages entre processus ,d'outils permettant la synchronisation de processus et éventuellement d'un gestionnaire d'écran.

Par comparaison avec les outils qu'offre UNIX,nous avons déterminé l'ensemble des outils que nous devions implémenter pour permettre la mise en oeuvre de l'application Petitpas. UNIX ne nous proposait pas un outil d'accès et de manipulation de fichier, de plus,les outils permettant la communication de messages entre processus et la synchronisation de processus ne nous parurent pas entièrement satisfaisants. Nous avons donc dû réaliser tous les outils que nous citons ci-avant.

Déroulement du travail.

La première étape du déroulement du travail a consisté en l'apprentissage du système UNIX,nous nous sommes limité dans un premier temps à une connaissance globale.

Ensuite,nous avons étudié le langage PASCAL,qui nous avait été imposé pour la réalisation de Petitpas.Afin de concrétiser nos connaissances théoriques de ce langage,nous avons transcrit une mini-application de gestion de commande et de stock écrite en BASIC par Mr.J.P. Adans.Cette mini-application étant surtout destinée à des démonstrations,elle n'est donc pas exempte de quelques imprécisions vis-à-vis des règles de gestion.

Cela nous a amené à écrire un rapport traitant des posssibilités et défauts du PASCAL pour le genre de problème qui nous préoccupe. Pour cela,nous avons comparé le PASCAL au langage BASIC.

La deuxième étape a consisté en un stage qui s'est déroulé à l'Unité d'Informatique de la Faculté des Sciences Appliquées à Louvain-La-Neuve.

Nous avons fait,alors,une étude des appels systèmes et des sous-routines de UNIX.Nous avons réalisé quelques programmes afin de se rendre compte des contraintes et des avantages de ces outils.

Ensuite,nous nous sommes attachés à l'analyse organique de Petitpas, c'est-à-dire au choix de l'architecture, à la spécification et à la conception des modules.

Cela nous a amené à définir les problèmes que nous allions rencontrer pour écrire Petitpas,à savoir :

- accès direct par clé aux fichiers,gestion de la concurrence de ces accès

- communication de messages inter-processus

- synchronisation de processus

Après plusieurs réunions de travail,nous avons pu esquisser les solutions à ces divers problèmes.

Notre première tche,fut de réaliser un outil qui allait permettre l'accès direct aux fichiers;cet outil étant,en effet,indispensable pour l'application considérée. Parallèlement à ce travail,nous avons affiné l'analyse organique du sujet.

De retour à Namur,nous nous sommes efforcés de mettre au point notre méthode d'accès,puis d'écrire les procédures

permettant la communication et la synchronisation entre processus.Ceci terminé,l'écriture complète des programmes de Petitpas put alors commencer.

CHAPITRE 2 : PRESENTATION DE UNIX.

Jean-Pierre Bodart

2. Présentation de UNIX[U5],[U8],[U9].

Remarque liminaire.

La présentation que nous faisons de Unix ne prétend pas être complète, loin de là; notre but en la faisant n'est pas de dire à nouveau ce qui a déjà été dit en long et en large, mais plutôt de présenter de Unix les outils principaux qui nous ont permis de réaliser notre travail. Pour une connaissance plus approfondie, le lecteur consultera les ouvrages renseignés dans la bibliographie.

Introduction.

Dans ce chapitre, après avoir rappelé les origines et quelques généralités de Unix, nous proposons une description en trois grands points : d'abord une présentation de l'organisation des fichiers; ensuite une description de certaines primitives de Unix qui permettent d'entrevoir les possibilités offertes par le système d'exploitation et qui surtout forment l'essentiel des matériaux que nous avons utilisés pour la réalisation de notre travail; enfin une brève mention de l'interpréteur des commandes de Unix qui constitue l'interface entre l'utilisateur et la machine.

2.0. Généralités.

Unix est un système d'exploitation écrit et conçu aux Bell Telephone Laboratories de type interactif et multi-utilisateur. Ce système d'exploitation a été écrit principalement pour des mini-ordinateurs de la gamme PDP11 de Digital Equipment Corporation. Ses grandes caractéristiques sont la souplesse, la simplicité et la puissance.

[U5] The UNIX Time-Sharing System
Dennis M. Ritchie and Ken Thompson
Bell Laboratories, Murray Hill, N.J. 07974

[U8] The UNIX Programmer's Manual
F.U.N.D.P (Namur)

[U9] Mesures de performances de UNIX sous une charge universitaire
(Mémoire présenté en vue de l'obtention du diplôme de licencié et maître en informatique.)
Jean-Paul Adans et Jean Vercheval
Institut d'Informatique (FUNDP), 1978-1979.

2.1. Organisation des fichiers.

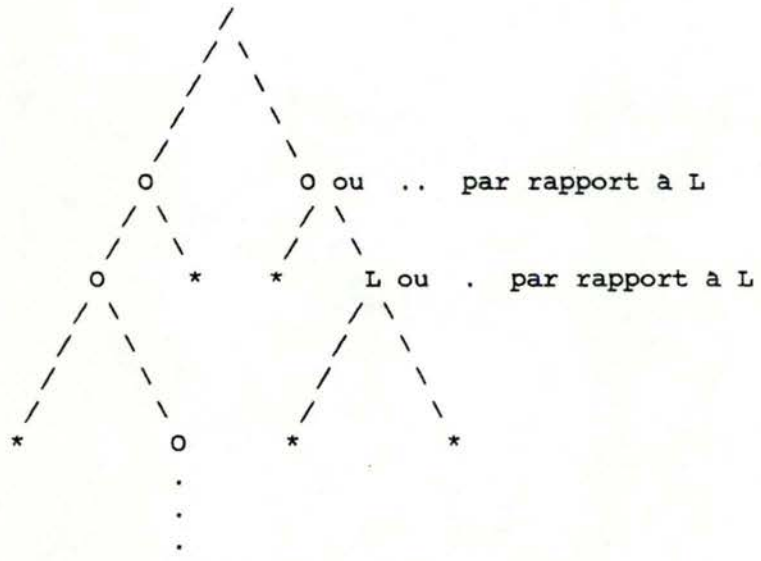
La structure de cette organisation est une arborescence dont les noeuds correspondent à des directoires, qui sont des éléments de chemin d'accès vers les fichiers qu'ils contiennent. Certains de ces directoires ont en plus la caractéristique d'être des points d'entrée dans le système pour les utilisateurs. Un point d'entrée (LOGIN) ou directoire par défaut correspond à un utilisateur et est le seul accès au système pour ce dernier; dès la connexion au système, ce point d'entrée constitue l'environnement initial de travail de l'utilisateur.

2.1.1. Classification des fichiers selon Unix.

Les fichiers Unix peuvent être classés selon trois types :

- directoires (ou annuaires ou catalogues) : constituent des chemins d'accès vers les fichiers de tout type qu'ils renferment,
- fichiers spéciaux : auxquels sont associés les périphériques,
- fichiers traditionnels.

Schéma.



* représente un fichier traditionnel,
O représente un directoire,
L représente un directoire point d'accès pour un nom d'utilisateur donné.

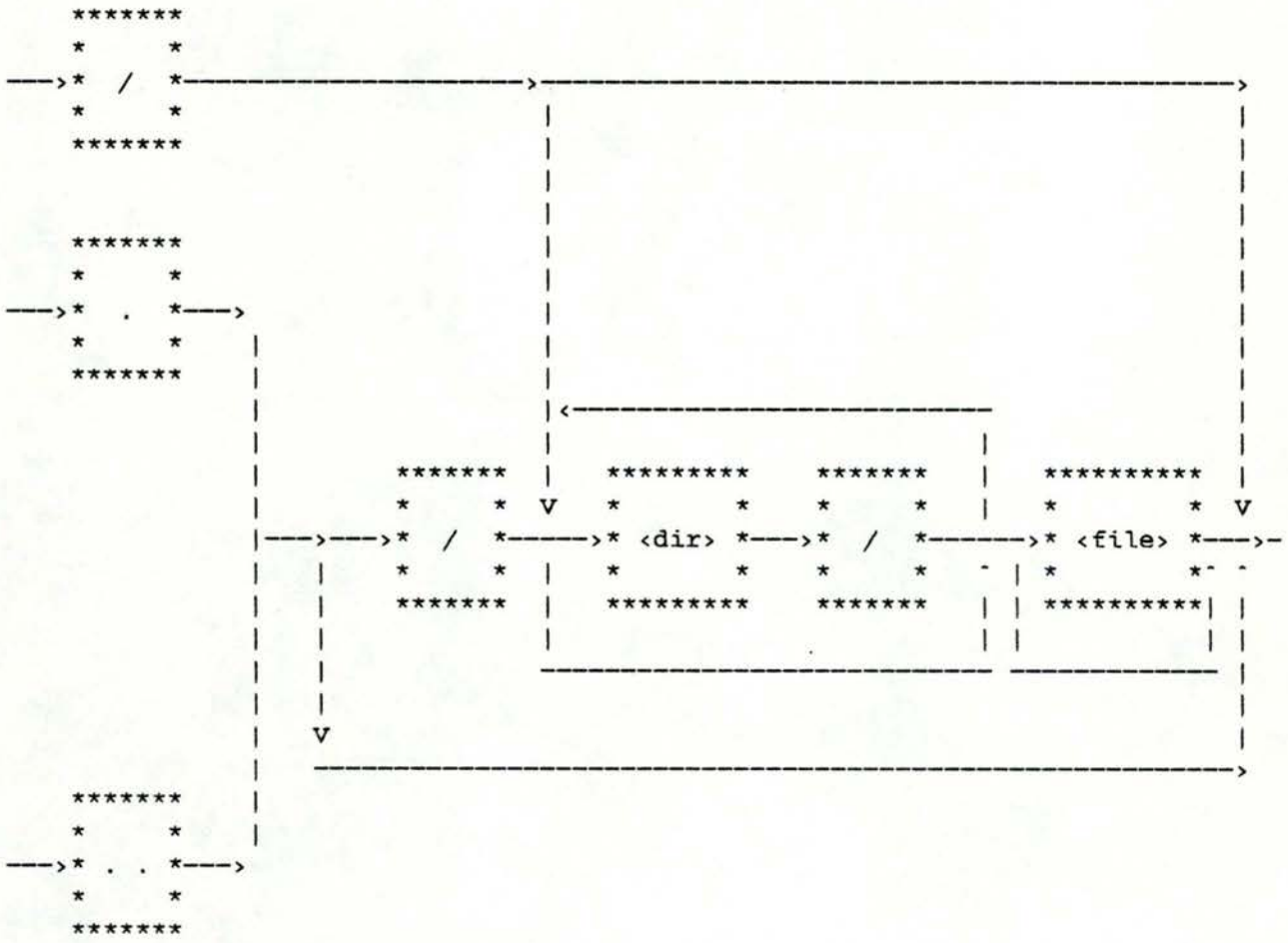
2.1.2. Désignation d'un fichier.

Considérons tout d'abord les trois notions fondamentales qui permettent d'identifier un fichier de l'arborescence et dont les symboles suivent :

- "/" : représente la racine de l'arborescence,
- "." : représente le directoire courant,
- ".." : représente le directoire parent.

Unix possède un principe de désignation d'un fichier quelconque qui consiste à donner une expression de chemin dans un catalogue; cette expression est appelée nom de passe.

En voici la syntaxe dans la représentation de Wirth :



<dir> : nom de directoire,
<file> : nom de fichier.

On remarque deux types d'accès :

- par nom absolu : référence à la racine,
- par nom relatif : référence au directoire courant ou au directoire parent.

2.1.3. Directoires.

Les directoires contiennent les correspondances entre les noms des fichiers et les fichiers eux-mêmes. De cette notion de directoire découle la structure arborescente du système des fichiers de Unix.

Unix permet de se déplacer dans l'arborescence, c'est-à-dire de changer de directoire (grâce à la commande chdir, [cfr U9]). Le changement de directoire respecte la protection d'accès aux fichiers.

2.1.4. Systèmes de fichiers amovibles.

Unix offre des systèmes de fichiers amovibles, car il n'est pas nécessaire que l'entièreté de la hiérarchie du système de fichiers soit en ligne à tout moment. Ainsi, le système de fichiers de Unix possède une partie montée en permanence contenant entre autre la racine de la hiérarchie du système et une autre partie contenant les systèmes démontables.
La gestion dynamique de ces systèmes amovibles est assurée par deux requêtes système de montage et de démontage de volume; un volume est un support physique de mémoire de masse. L'effet de la requête de montage est de substituer une feuille de l'arbre hiérarchique du système résident par le sous-arbre correspondant contenu dans le volume amovible; la requête de démontage possède l'effet inverse.

2.1.5. Protection des fichiers.

Classes d'utilisateurs.

Unix introduit les notions :

- de super-utilisateur : utilisateur privilégié du système qui dispose de tous les droits,
- de propriétaire : lequel est le possesseur du fichier; par défaut celui-ci est le directoire "par défaut" contenant ce fichier, mais le propriétaire peut être changé par le super-utilisateur,
- de groupe d'utilisateurs : ensemble d'utilisateurs privilégiés dont fait partie le propriétaire,
- d'utilisateur quelconque : tout utilisateur ne faisant pas partie d'un groupe dont fait partie le propriétaire.

Types d'accès.

- en écriture,
- en lecture,
- en exécution : en ce qui concerne un catalogue, le droit d'accès en exécution doit être interprété comme le droit de parcourir le sous-arbre défini à partir de ce catalogue.

Droits d'accès.

Pour chacune des classes d'utilisateurs définies ci-dessus, le propriétaire du fichier dresse la liste des droits d'accès, obligatoirement inclus dans l'ensemble écriture, lecture et exécution.

2.1.6. Structure des fichiers.

Un fichier est considéré par Unix comme une suite de caractères; Unix ne reconnaît aucune structure sur ceux-ci. C'est à l'utilisateur de définir ses propres structures et de les gérer par lui-même.

2.1.7. Méthodes d'accès aux fichiers.

Le système Unix n'associant aucune structure aux fichiers, sinon celle de suite de caractères, il ne fournit, par conséquent, pas de méthode classique d'accès, telles les méthodes séquentielle, indexée séquentielle ou aléatoire. Ceci étant, Unix met à la disposition du programmeur trois primitives permettant de manipuler un fichier et qui sont :

- "READ"

  Forme de la primitive.

  nb_car_lus = read(descripteur_de_fichier, tampon, nb_car_a_lire);

  Fonction.

  La primitve "read" permet de demander la lecture de "nb_car_a_lire" caractères d'un fichier de descripteur "descripteur_de_fichier" dans une zone "tampon" à partir de la position courante dans le fichier. La primitive retourne le nombre de caractères effectivement lus dans "nb_car_lus".

- "WRITE" : idem que "read", mais en écriture.

- "SEEK"

  Forme de la primitive.

  position = seek(descri_fich, depl, base);

  Fonction.

  La primitive "seek" permet de se déplacer en avant ou en arrière d'un nombre "depl" de caractères, dans le fichier de descripteur "descrip_fich", relativement au début, à la fin ou à la position courante dans le fichier selon ce que précise le paramètre "base". La nouvelle position courante dans le fichier exprimée par rapport au début de celui-ci est disponible dans la zone position.

A ces primitives de manipulation du contenu d'un fichier, s'ajoutent naturellement des primitives de création, ouverture et destruction d'un fichier.

2.2. Primitives du système Unix.

Introduction.

Unix offre un nombre important de primitives ou fonctions du système disponibles à deux niveaux; à celui du programmeur et à celui de l'utilisateur au terminal. Nous avons décidé de ne présenter qu'une partie seulement de ces primitives parce qu'elles constituent les matériaux qui ont permis de construire le système d'informations de Petitpas.
Ce sous-ensemble de primitives, nous l'avons découpé en quatre grandes classes :

- les primitives manipulant les caractéristiques déterminant l'état d'un fichier,
- celles permettant la manipulation de processus,
- celles permettant la communication de données entre processus,
- celles, enfin, permettant de synchroniser des processus.

2.2.1. Manipulation d'un fichier et de son état.

Remarque liminaire.

Une contrainte sérieuse imposée par Unix est que par processus, il ne peut y avoir que 15 fichiers ouverts. Ceci est dû à l'implantation du système qui associe à chaque processus une table des fichiers ouverts, dont la taille est fixe. L'utilité de cette table est de servir de relais d'adressage des fichiers pour augmenter la rapidité d'accès.

2.2.1.1. Création d'un fichier.

Forme de l'appel.

descripteur = creat(nom_de_fichier, mode).

Fonction.

La primitive "creat" permet la création d'un fichier de nom "nom_de_fichier" selon le mode d'accès "mode". Celui-ci détermine le type d'accès possible et leurs utilisateurs [cfr 2.1.5.]. Le fichier est initialement vide ou s'il existait déjà il est ramené à la longueur nulle. Si ce fichier a pu être créé, un numéro de la table des fichiers ouverts lui est assigné et est renvoyé dans "descripteur". Toute manipulation ultérieure du fichier se fera au moyen de ce descripteur.

2.2.1.2 Création d'un lien de fichier.

Forme de l'appel.

link(nom1, nom2).

Fonction.

La primitive "link" attribue un synonyme "nom1" au fichier existant de nom "nom2". L'attribution d'un lien n'est possible qu'à l'intérieur d'un même volume [cfr U5 §3.4].

2.2.1.3. Suppression d'un lien de fichier.

Forme de l'appel.

unlink(nom).

Fonction.

La primitive "unlink" permet de détruire un lien de fichier et donc dans le cas où celui-ci est le seul ou le dernier lien du fichier, il y a destruction effective du fichier de nom "nom".

2.2.2. Manipulation de processus.

2.2.2.0. Définitions.

a) Définition d'un processus.

Unix considère un processus comme étant l'exécution d'une image de programme. L'image constitue l'environnement d'exécution d'un programme comprenant :

- la copie du code objet du programme en exécution, c'est-à-dire contenu en mémoire,
- la valeur des registres généraux,
- le directoire de travail,
- les fichiers ouverts,
- les relations avec d'autres processus ("pipes", etc ...).

Remarque.

Unix associe à tout processus un numéro que l'on qualifie d'identificateur de processus.

b) Definitions de processus "père" ....

Un processus "père" est un processus qui crée un autre processus appelé "fils" au moyen de la primitive "fork". Les termes "parent" et "enfant" sont une nuance de langage évoquant le fait que le texte du processus "père" et/ou celui du processus "fils" a changé [cfr 2.2.3.2]. Ces différents termes se rapportent tous à une et même notion fondamentale "ascendant-descendant" de Unix. Nous utiliseront parfois les termes de processus générateur en synonyme de père et de processus généré pour fils.

2.2.2.1. Création d'un nouveau processus.

Forme de l'appel.

numéro_de_processus = fork( ).

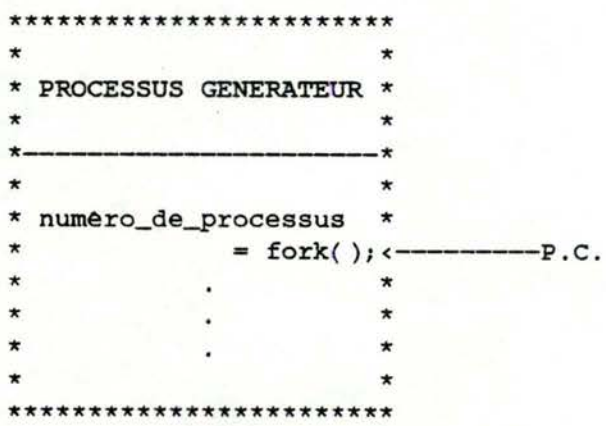
Fonction.

La primitive "fork" permet de créer un nouveau processus "enfant" à partir d'un processus "père". Ce nouveau processus est la copie de celui qui l'a créé, en effet, les deux processus résultant du dédoublement possèdent la même copie de la mémoire à une différence près cependant, et de laquelle découle en partie l'intérêt de cette primitive, à savoir que "fork" renvoit au processus père le numéro du processus fils et au fils, il retourne la valeur nulle. C'est de cette différence que découle l'intérêt de cette primitive puisqu'elle permet dans le corps du programme de tester la valeur de "numéro_de_processus" et de dédicacer une fonction particulière à chacun des processus.

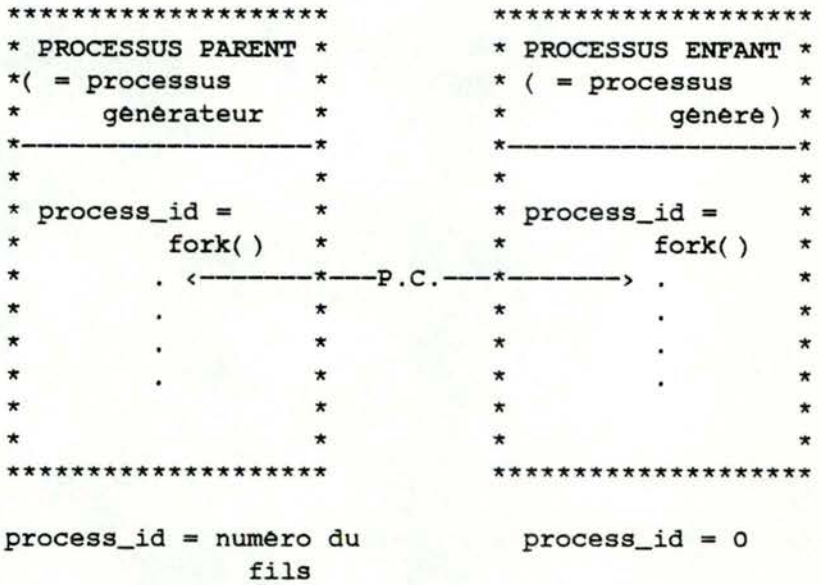
Le processus fils commencera son exécution où celle du père reprendra son déroulement, c'est-à-dire après l'appel à "fork".

Schèma.

Avant l'exécution de "fork".



Après l'exécution de "fork".



process_id = numèro du fils

process_id = 0

Dans les deux cas, le compteur de programmes (P.C.) relatif à chaque processus pointe l'instruction suivant l'instruction "fork".

2.2.2.2. Substitution d'un processus.

Forme de l'appel.

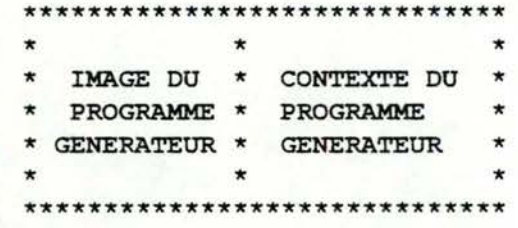
execute(fichier, arg1, ..., argn).

Fonction.

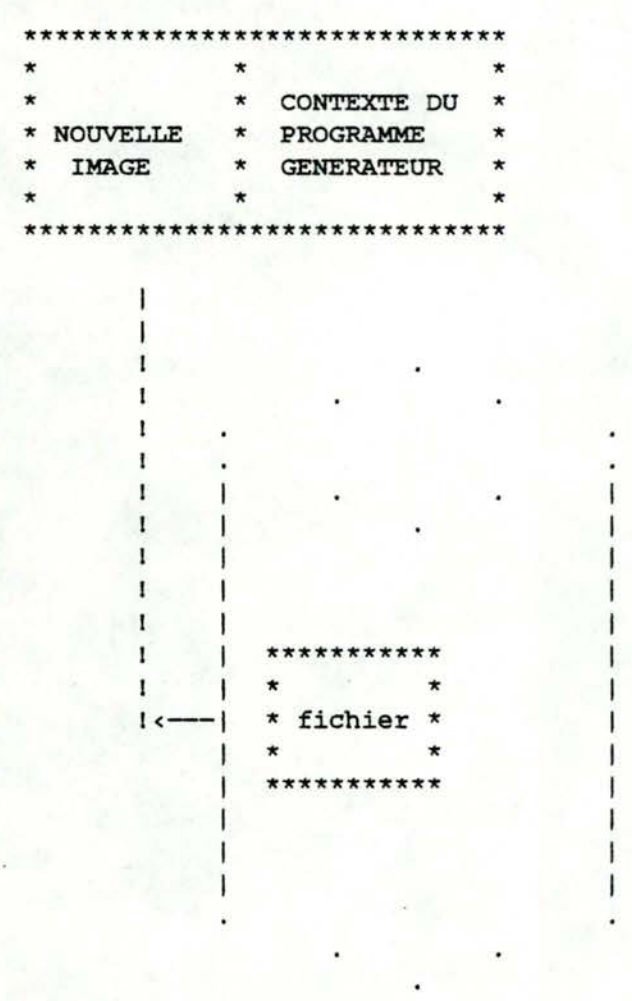
La primitive "execute" permet d'exécuter un programme se trouvant dans un fichier en lui donnant les paramètres nécessaires . Contrairement à la primitive "fork", il n'y a pas création d'un nouveau processus, mais bien une substitution. Le code objet et les données du programme à exécuter recouvrent l'espace du code et des données du processus qui a exécuté cet ordre. Le nouveau processus hérite de l'état du processus générateur, ainsi que des fichiers ouverts et des relations avec d'autres processus.

Schéma.

Avant "execute".



Après "execute".



2.2.2.3. Terminaison d'un processus.

Forme de l'appel.

exit(état_du_fils).

Fonction.

La primitive "exit" provoque la terminaison d'un processus et la fermeture de tous les fichiers ouverts ouverts par celui-ci. Elle provoque la reprise du déroulement du processus parent qui aurait exécuté un "wait" [cfr 2.2.4.2.]; ce dernier peut disposer alors de l'"état_du_fils".
Elle ferme également tous les fichiers du processus.

2.2.3. Communication de données entre processus.

2.2.3.1. Création d'un canal inter-processus.

Forme de l'appel.

pipe(paire_de_descripteurs).

Fonction.

La primitve "pipe" permet à un processus de créer un moyen de communication avec un ou plusieurs autres processus. Ce moyen de communication ou "canal" est vu par chacun des processus comme un fichier et doit être créé par un seul processus. Après l'appel de la primitive, les deux paramètres contiennent les valeurs des descripteurs associés à ce fichier; l'un d'eux sert à l'écriture et l'autre à la lecture. Chacun des processus peut utiliser les deux descripteurs. Un canal est un moyen de communication géré selon la méthode FIFO. Unix ne permet la création de canaux qu'entre un processus parent et des processus enfant [cfr 2.2.2.1.].

2.2.3.2. Envoi et lecture des informations contenues dans un canal.

Comme pour les fichiers habituels, les primitives "read" et "write" permettent la lecture et l'écriture d'informations transmises dans un canal.

2.2.4. Synchronisation des processus.

Définition d'un signal.

Un signal est une ressource de Unix représentant un évènement.

2.2.4.1. Annonce d'un évènement.

Forme de l'appel.

kill(identificateur, type_de_signal).

Fonction.

La primitive système "kill" permet de positionner un type de signal implanté dans le système correspondant à un évènement défini.

2.2.4.2. Attente d'un évènement.

Forme de l'appel système.

numéro_de_processus = wait(status).

Fonction.

"Wait" permet à un processus père de suspendre le déroulement de son exécution dans l'attente de la terminaison d'un processus fils. Il n'est pas possible au processus père de préciser le processus fils dont il attend la clôture d'exécution, ainsi dans le cas où un processus père devrait attendre la terminaison de plusieurs processus enfants, autant d'appels à la primitive "wait" que de processus enfants impliqués devraient être explicitement formulés.

Dans numéro_de_processus, on trouvera le numéro du processus.

2.2.4.3. Analyse d'un évènement.

Forme de l'appel système.

signal(type_de_signal, fonction).

Fonction

La primitive "signal" permet au processus récepteur d'un signal de spécifier la routine "fonction" qui doit être exécutée lors de la réception du signal précisé par "type_de_signal". L'action de "fonction" peut être soit d'ignorer le signal, soit d'appeler une routine de service de l'interruption, soit de réagir normalement, c'est-à-dire

par un "exit"[cfr 2.2.3.3.].

Remarque.

L'inconvénient de ces primitives est l'absence de file d'attente devant un signal.

2.3. Interpréteur de commandes : le Shell.

Le Shell est un programme interpréteur de commandes enjointes par tout utilisateur au terminal. Son rôle ne se limite pas à l'interprétation seule, mais aussi à l'exécution de la commande.

2.3.1. Commandes disponibles.

Les commandes disponibles sont nombreuses; nous pouvons citer par exemple des compilateurs, des éditeurs de texte, des formateurs de texte, et bien d'autres commandes dont certaines répondent à la classification des primitives présentée dans le point 2.2.

2.3.2. Indirection.

A toute commande traitée(interprétée + exécutée) par le Shell sont associés deux fichiers correspondant aux flux d'informations d'entrée et de sortie de la commande; l'indirection offre toute liberté à l'utilisateur d'aiguiller à sa guise les flux d'entrée et de sortie d'une commande du Shell. Par défaut, ce sont les flux standards (en entrée le clavier et en sortie l'écran).
Ces trois opérateurs sont :

- "<" : permet de renseigner un autre flux d'entrée que le flux standard d'entrée,
- ">" : permet de renseigner un autre flux de sortie que le flux standard. Le flux de sortie recouvre dans ce cas le contenu du fichier renseigné comme récepteur,
- ">>" : le flux de sortie est dirigé vers le fichier renseigné, mais est ajouté à la suite du contenu existant.

2.3.3. Programmation du Shell.

Le Shell offre la particularité d'être programmable; en effet, il existe des ordres de contrôle tels : if, goto, etc ... qui permettent de construire des programmes de commandes. Nous nous contentons de mentionner la chose sans entrer plus avant dans la présentation.

CHAPITRE 3 : PRESENTATION DE PETITPAS.

Jean-Paul Tixhon

3. Description de Petitpas. [A12]

Pour ce mémoire,nous devions étudier les possibilités du système UNIX en informatique de gestion,il était donc utile de réaliser une application de gestion sur ce système.Le choix s'est tourné vers le problème de la firme Petitpas.

Deux raisons nous ont dicté ce choix,d'une part,le fait que cette application est très réaliste et donc représentative d'un problème que l'on pourrait rencontrer dans une firme,et d'autre part, le fait que l'analyse de cette application avait déjà été abordée dans le cadre de nos études.

Dans ce chapitre,nous allons donc présenter de manière détaillée l'application Petitpas,avant et après l'informatisation,et en donner les caractéristiques fondamentales,qui nous permettront de voir ce qui est nécessaire à un système d'exploitation pour pouvoir supporter ce type d'application.

3.1 Introduction au problème.

Pour commencer,exposons la situation générale de la firme Petitpas.Cette firme est une entreprise de vente par correspondance, imaginaire,mais que l'on pourrait comparer à une entreprise telle que La Redoute.

Le problème fut en fait proposé par les membres de l'unité de gestion de l'Institut d'informatique dans le but d'appliquer à un cas représentatif d'une firme désirant s'adjoindre l'outil informatique,une démarche d'analyse développée à l'Institut et enseignée aux étudiants.

Petitpas vend des vêtements par correspondance,ces vêtements sont achetés à diverses firmes de fabrication textile et l'entreprise n'a donc aucune activité de production,étant ainsi un agent intermédiaire.

L'entreprise envoie à ses clients,anciens ou potentiels,deux catalogues par an,un catalogue pour sa collection de vêtements d'été et l'autre pour sa collection d'hiver.Les acheteurs renvoient par la poste les bons de commande,qui sont fournis avec les catalogues,à la firme. Celle-ci tente alors de satisfaire ces commandes et elle livre ses produits par l'intermédiaire des réseaux de distribution de la poste et de la SNCB.

Dans les chapitres suivants,nous reviendrons dans le détail sur ces procédures, en présentant l'organisation de la firme et la solution automatisée choisie.

Mais avant,il est nécessaire d'expliquer les raisons qui ont poussé les responsables de l'entreprise à tenter l'informatisation de certaines activités de gestion.

L'objectif de gestion fondamental est d'améliorer la rentabilité financière à court terme.En effet,comme la plupart des entreprises,et particulièrement les entreprises de vente par correspondance où les marges bénéficiaires sont réduites,il est important que le rendement de l'argent soit optimal.

Cet objectif fondamental implique trois sous-objectifs :

a) l'amélioration de la gestion des stocks.

Il est d'une part nécessaire de diminuer la quantité de produit en stock,car le coût de détention est important,et d'autre part d'éviter les ruptures de stock pour ces produits. Les responsables devrons donc trouver un point d'équilibre entre ces deux objectifs,et réaliser un suivi précis de la quantité de produits disponibles.L'outil informatique permettra certainement d'améliorer cette connaissance des stocks,notamment grâce à l'adoption d'une gestion en temps réel.

b) l'amélioration des rentrées financières,la réduction du découvert financier.

Cet objectif intéresse le département financier.

c) la réduction des coûts,d'une part en ce qui concerne les commandes différées des clients,et d'autre part en ce qui concerne la gestion administrative.

On entend par commande différée,une commande qui ne peut être satisfaite en une seule expédition.

En réduisant le nombre de commandes de ce type,grâce à une gestion automatisée et à une connaissance en temps réel du stock,l'informatique peut en diminuer les coûts.

Les autres coûts touchent la gestion administrative de la firme, or depuis le début de son existence,l'informatique de gestion a souvent été utilisée pour réduire les procédures administratives ou,au moins, pour améliorer la rapidité de ces procédures,donc pour diminuer leurs prix.

On attend donc une nette amélioration dans ce secteur.

Un second objectif de gestion est d'améliorer le service à la clientèle et l'on remarque que la réalisation des différents sous-objectifs précédents,améliorera en partie cet objectif désiré par le département du marketing.

De ces objectifs de gestion,nous pouvons rapidement déduire quelques objectifs purement informationnels(et pas seulement informatiques).

Pour le premier objectif(améliorer la rentabilité financière),nous avons :

-premier sous-objectif (gestion des stocks) :

+ réduction du délai d'enregistrement d'une commande client

+ réduction au minimum du délai d'enregistrement d'une livraison fournisseur au niveau du stock

+ réduction au minimum du temps de diffusion de l'information concernant le stock(gestion en temps réel)

+ meilleur définition des paramètres de gestion de stock

+ meilleure connaissance des ventes(statistiques)

+ meilleur connaissances des fournisseurs(réduire le délai du choix,améliorer ce choix)

-deuxième sous-objectif (amélioration des rentrées financières) :

+ accélération du traitement des factures

+ réduction du temps de séjour d'une commande client dans la firme

+ meilleure gestion des clients(suivi financier)

+ obtention de meilleures conditions de paiement des fournisseurs

-troisième sous-objectif (réduction des coûts) :

+ réduction du nombre de rupture

+ diminution du coût de rupture

Pour le second objectif(service clientèle),nous avons :

+ réduction du délai de livraison

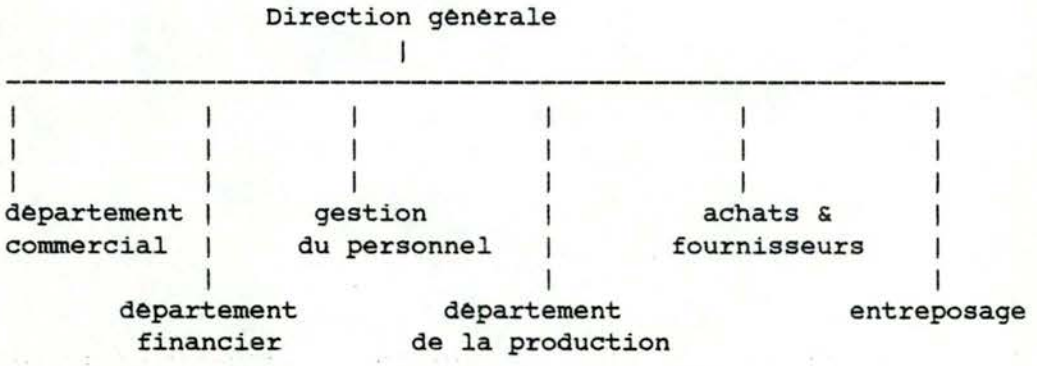
+ réduction du nombre de différés et de commandes refusées

On notera que ces objectifs ne seront pas tous atteints grâce à l'informatisation.

3.2 Présentation de l'organisation de la firme.

Après avoir présenté le problème Petitpas, nous allons dans un second temps expliquer dans le détail l'organisation de l'entreprise et les rôles que remplissent les départements,et ceci avant l'informatisation.

L'organigrammme de Petitpas est le suivant :



Quelles sont les f nctions et la structure interne de ces départements ?

Département financier.

Ce département a pour but de gérer les finances de l'entreprise,il sera peu touché par l'informatisation.

Département de gestion du personnel.

Ce département s'occupe de la gestion du personnel(recrutement, licenciement, promotion , salaire, etc..), comme le précédent, nous ne nous y attarderons pas.

Département commercial.

Ce département qui s'occupe du marketing de la firme,a pour fonction principale d'assurer la parution et la réalisation des catalogues présentés par Petitpas.Le service assure également,le suivi des produits,à partir des statistiques,pour déterminer les options futures de l'entreprise. Le type de produit à promouvoir est donc défini par ce service.

De plus,ce service effectue la gestion de la clientèle. Sa tâche est de définir les clients potentiels,d'effectuer le suivi des clients,de mettre en place les politiques de relance et de contacts avec ceux-ci.Il est aussi de sa compétence de prévoir l'augmentation

annuelle du nombre de clients par marchè,par produit...

Dèpartement achats et fournisseurs.

Ce dèpartement a pour responsabilitè d'effectuer la gestion des fournisseurs : acceptation de nouveaux,suppression d'anciens,choix des fournisseurs pour une commande,envoi et suivi de ces commandes,enregistrement des livraisons,analyse de l'ètat du stock des produits.

Dèpartement d'entreposage.

Ce dèpartement assume la rèception physique des livraisons fournisseurs,du contrôle des marchandises(quantitatif et qualitatif). Il est ègalement responsable du rangement dans les magasins de ces produits.

De plus,ce service s'occupe des retours de produits livrès à un client non satisfait.

Dèpartement de la production.

Ce service traite les commandes clients.

Il est divisè en quatre sous-services : enregistrement,prèparation des colis,facturation,expèdition.

Enregistrement

Le but de ce service est de contrôler et d'enregistrer les commandes clients.Après un contrôle sur l'identitè du client,la vèrificatrice contrôle le bon de commande.Ce dernier n'est enregistrè que si toute ses lignes sont correctes,sinon le bon fera l'objet d'un traitement d'exception pour tenter de rècupèrer certaines erreurs(c'est ce que nous appelerons le traitement des cas litigieux)dans le but de limiter au maximum le nombre de commandes refusèes.

Prèparation des colis

Ce service assure la prèparation des colis. Il recherche dans le stock les produits et forme les colis à expèdier.

Facturation

Ce service est chargè de rèaliser la facture associèe à une livraison et èventuellement de remplir un justificatif de non livraison pour les commandes diffèrèes.Un bordereau d'expèdition,pour la poste ou la SNCB,est ègalement fourni avec la facture.

Expédition

Lorsque les documents d'accompagnement arrivent dans ce service en provenance de la facturation, les manutentionnaires se les répartissent et recherchent les colis. Ils assurent ensuite le tri par mode d'expédition, SNCB ou poste, et l'expédition des colis.

De plus, certains employés sont chargés de la mise à jour des fichiers des stocks, après les livraisons des fournisseurs ou l'expédition des colis. Cette mise à jour est quotidienne, ces employés envoient l'état des stocks aux services compétents.

3.3 Présentation de la solution automatisée.

Cette proposition d'automatisation présente le futur système informatique que l'on pense installer dans la firme Petitpas (ce texte découle du document "Proposition d'automatisation"[A12]).

Elle reprend la nouvelle structure de l'entreprise ,les nouveaux traitements appliqués aux commandes clients et aux livraisons fournisseurs, les traitements qui seront en priorité informatisés.

Quelle sera la nouvelle organisation de la firme ?

1. Les départements commercial,financier,de gestion du personnel,achats et fournisseurs restent inchangés.Ils seront en effet peu touchés par l'informatisation.

2.Le département production sera organisé comme suit :

2.1 Le service d'enregistrement des commandes procédera à l'aide de terminaux

- au contrôle des commandes clients
- à la mise à jour du fichier des adresses clients
- à l'enregistrement des commandes clients

2.2 Le service de préparation des commandes effectuera des recherches par série

2.3 Le service emballage et expédition procédera à l'aide de terminaux

- à la reconstitution de la commande
- au lancement de la facturation

Il s'occupera,de plus,de l'emballage et de l'expédition des colis.

2.4 Inventaire permanent

Tous les vendredi midi,les services de préparation des commandes,d'emballage et d'expédition cesseront leur travail habituel et procéderont à un inventaire permanent.Ils rangeront les rayons,comptabiliseront le nombre d'articles de chaque rayon et signaleront l'état des stocks à un gestionnaire qui corrigera éventuellement la quantité en stock dans le fichier d'état des stocks.

3. Le département d'entreposage procédera à l'aide de terminaux au controle et à l'enregistrement des livraisons fournisseurs.

Les services concernés sont reliés à l'ordinateur central(dans notre cas)par une liaison en temps réel,qui gérera les ressources du système(mise-à-jour du stock,gestion des commandes clients,des livraisons fournisseurs,etc...).

Quelles seront les nouvelles procédures ?

1.Enregistrement et contrôle des commandes clients.

1.1 Préparation du bon de commande.

A chaque arrivée de courrier,ce poste reçoit les bons de commandes ,il décachète les enveloppes et s'assure que les bons sont signés.Les bons non signés,sont transmis à la cellule des cas litigieux,les autres continuent la filière.

1.2 Controle du bon de commande.

Ce poste est composé de quelques opératrices qui sont en liaison directe avec l'ordinateur grâce à un terminal.En cas d'échec,le bon de commande est transmis à la cellule ds cas litigieux.

L'opératrice effectue dans l'ordre les opérations suivantes :

Identification du client.

On distingue deux cas ,selon que le bon possède un numéro de client ou que le bon ne possède pas de numéro préimprimé.

Dans le premier cas,l'opératrice procédera à une recherche dans le fichier client pour vérifier l'existence de ce numéro et la concordance du nom si ce numéro existe.Si le numéro n'existe pas,elle effectuera une recherche sur les nom,prénom,adresse afin de vérifier une concordance éventuelle.

Dans le second cas,l'opératrice procédera à une recherche sur nom,prénom et adresse.

Constitution du corps de la commande.

L'opératrice introduit la commande ligne par ligne.Elle a pour objectif d'interpréter au mieux le bon de commande du client afin de perdre le moins de commande possible.Pourtant si elle ne peut interpréter une ligne elle abandonne entièrement la commande.

La règle d'interprétation est d'accorder la priorité au libellé du produit par rapport au numéro de

référence. L'opératrice procède à l'identification d'une ligne par une analyse de concordance de libellé.Pour un produit identifié,en l'absence d'une quantité commandée mentionnée par le client,l'ordinateur prendra la quantité par défaut.

Si la commande a du être abandonnée au cours du traitement d'une de ses lignes,l'opératrice marque la ligne litigieuse sur le bon de commande et expédie celui-ci à la cellule de traitement des cas litigieux.

Si toutes les lignes du bon de commande ont pu être interprétées, l'ordinateur affiche alors le prix total à payer par le client. L'écart entre le total calculé et celui du bon de commande est à apprécier par l'opératrice,qui d'cide d'enregistrer ou de refuser la commande.

1.3 Traitement des cas litigeux.

Ce bureau examine les bons de commandes refusés et tente de les recycler.A défaut,il envoye au client son bon de commande annoté,un justificatif de refus et un nouveau bon de commande.

2. Préparation et enregistrement des commandes aux fournisseurs.

Quotidiennement,le service des achats aux fournisseurs consulte le fichier d'état des stocks.Cette interrogation fournit une liste des produits à réapprovisionner sur base des calculs suivants :

Niveau des ressources =
Quantité en stock +
Quantité en commande chez les fournisseurs -
Quantité due pour les commandes différées.

Si le niveau des ressources est < ou = au point de commande,il y a décision de réapprovisionnement.La quantité à commander doit être le plus petit multiple de la quantité économique de commande supérieur à la différence entre le point de commande et le niveau des ressources.Connaissant ces informations,le service des achats consulte éventuellement le fichier des fournisseurs et celui des commandes en cours,contacte les fournisseurs et établit un bon de commande.

Par fournisseur et par produit,le gestionnaire des achats possède les renseignements suivants (relatifs aux produits) :

- prix unitaire
- délai de livraison
- capacité de livraison
- pourcentage de remise
- coûts de transport

3. Réception de la commande fournisseur.

Le département d'entreposage réceptionne les livraisons en provenance des différents fournisseurs.Il effectue le contrôle des marchandises:

- en consultant en temps réel le fichier des commandes fournisseurs en cours,il vérifie le fait que la livraison correspond en tout ou en partie à une commande chez ce fournisseur,

- en s'assurant que la marchandise est conforme aux normes de qualité exigées.

4. Mise à jour de l'état des stocks.

4.1 Traitement des entrées en stock(mise à jour plus).

Ce traitement est activé chaque fois qu'une transaction "produit réapprovisionné" est créé par le traitement de réception des commandes fournisseurs.

La quantité emmagasinée est ajoutée à la quantité disponible en stock et soustraite de la quantité en commande auprès du fournisseur.

Après la mise à jour des quantités en stock,l'ordinateur traitera les commandes différées.Le système devra sélectionner les commandes dont une ligne au moins est relative à un produit réapprovisionné.Ces commandes seront ensuite traitées par la mise à jour moins du stock,comme pour les commandes normales.

4.2 Traitement des sorties de stock(mise à jour moins) et ordonnancement.

Ce traitement est activé par deux types de transactions :

- une commande enregistrée par le service d'enregistrement des commandes
- une commande différée sélectionnée.

Les commandes différées sélectionnées sont traitées en priorité par rapport aux commandes normales.

Pour une commande enregistrée,par ligne on effectue les opérations suivantes :

- si le produit est supprimé,on indique ce fait pour cette ligne de commande

- si le stock n'est pas épuisé et est suffisant pour satisfaire la demande, on livre ce qui est demandé(avec mise à jour du stock)

- si le stock n'est pas épuisé mais non suffisant à l'égard de la demande, on livre la quantité en stock(avec mise à zéro du stock), la ligne devient une ligne de commande différée,il reste encore une quantité à livrer

- si le stock est à zéro,on ne livre rien et la ligne devient une ligne de commande différée.

Pour une commande sélectionnée,les règles sont très proches.

Pour chaque produit commandé,l'existence d'une quantité à livrer engendre une réquisition.On notera que si un produit est épuisé,un message est envoyé au service de facturation.

5.Préparation des colis.

Les bons de réquisition par commande,émis par le traitement de mise à jour moins,sont accumulés pour être groupés en 100 commandes , ce qui permet au magasinier de préparer ces 100 commandes en optimisant son parcours au travers des rayons.

Pour ranger les commandes,le magasinier dispose d'un chariot divisé en 10 casiers dans lesquels il dispose les marchandises correspondant à 10 commandes.

En plus du bon de réquisition par série,le magasinier reçoit un bon par casier,qui spécifie les 10 commandes affectées à ce casier.Ce bon par casier,il l'agraphe au casier correspondant lorsqu'il commence son parcours en magasin.

6.Emballage et facturation.

Le service d'emballage se compose de quelques stations équipées ,au point de vue informatique, chacune d'un terminal à écran et d'une imprimante - une seule imprimante pour toutes les stations - capable de lister tous les documents commandés par les difffrentes stations d'emballage.

Au fur et à mesure de l'arrivée des chariots,les préposés,à l'aide du bon par casier,connaissent les numéros des commandes préparées dans un casier.Chaque préposé,responsable d'une station d'emballage,affiche à l'écran la commande qu'il a décidé de traiter.

En puisant dans le casier et en suivant à l'écran,il reconstitue le colis.S'il peut reconstituer ce colis

entièrement,il enclenche la facturation c'est-à-dire :

- l'impression d'une facture et d'un justificatif de non livraison éventuel

- l'impression d'une étiquette de colisage

- l'impression d'un bordereau SNCB ou poste suivant le poids total du colis.

Cette facturation génère un mouvement comptable qui ira s'archiver sur le fichier des mouvements comptables.

S'il ne peut reconstituer le colis entièrement,il met momentanément cette commande en suspens et poursuit avec les autres commandes du casier.

Quand toutes les commandes du casier ont été passées en revue,il prend en charge la ou les commandes restées en suspens.Un magasinier, travaillant sous ses ordres,replace dans les rayons les marchandises restées dans le casier et essaye de compléter la ou les commandes incomplètes.

Si malgré ces recherches,le colis reste incomplet,il décide de l'envoyer tel quel au client mais corrige à l'écran les quantités réellement livrées.Après cette correction,il effectue une série d'opération :

- effectuer une mise à jour du stock pour le produit incompletement livré

- effectuer une mise à jour du fichier des commandes diffrées (ajoût d'un du)

- avertir par téléphone,le gestionnaire des stocks,que certains stocks sont inconsistants.

3.4 Caractèristiques informatiques de l'application Petitpas.

Puisque l'application Petitpas doit être rèalisèe sur le système UNIX,nous devons tenter de dègager les caractèristiques fondamentales de cette application,pour les comparer aux possibilitès de UNIX;et ainsi, mettre le doigt sur les insuffisances de ce système d'exploitation.

Pour notre mèmoire,nous nous situons dans le cadre d'un environnement informatique centralisè,non distribuè,nous rèaliserons donc l'application sur un mini-ordinateur auquel seront raccordès des terminaux non intelligents.

On soumettra au système d'exploitation des travaux de type administratif et de gestion.Il doit permettre une charge transactionnelle importante,ainsi qu'une charge batch(pour le dèroulement d'applications futures).Il est indispensable que le système soit multi-utilisateurs et puisse supporter plusieurs vidèos.Il doit ègalement permettre la multi-programmation et le multi-tâche.

Quelles sont les caractèristiques nècessaires du logiciel de dèveloppement , que l'on peut prèvoir à ce stade ?

Les langages :

Nous devons pouvoir disposer d'un langage dit de gestion (exemples : COBOL,RPG,PL/1,BASIC).Nous utiliserons,nous,le langage PASCAL, et ce,pour dèmontrer ses possibilitès et ses dèfauts dans ce domaine.

Les utilitaires ,les primitives :

Nous devons pouvoir disposer des primitives ou des utilitaires suivants :

- communication entre processus : il est indispensable de pouvoir envoyer des messages entre des programmes totalement indèpendants. Par exemple,entre le programme d'enregistrement des commandes et le programme de mise à jour moins du stock.

- gestionnaire d'ècran : il serait souhaitable de disposer de cet utilitaire,pour rèaliser facilement un dialogue agrèable avec l'utilisateur.

- gestionnaire de fichiers permettant :

l'accès sèquentiel, l'accès direct par clè identifiante ou non identifiante,numèrique ou alphanumèrique

la gestion des accès en concurrence : de nombreux

fichiers de Petitpas étant accédé par plusieurs programmes(exemple : le fichier des commandes clients).

la protection d'accès aux données : les fichiers ne peuvent être atteints que par les programmes de l'application, ces mêmes programmes ne peuvent accéder qu'à certains fichiers.

Citons encore,de manière non exhaustive,quelques outils nécessaires :

- un éditeur de texte

- un éditeur de lien

- un gestionnaire de librairie.

CHAPITRE 4 : ANALYSE FONCTIONNELLE.

Jean-Paul Tixhon

4. Analyse fonctionnelle. [A2] [A3] [A4] [A5] [A6]

4.1 But et méthode de l'analyse fonctionnelle.

But de l'analyse fonctionnelle.

L'objectif de ce chapitre est de donner un aperçu global de l'analyse fonctionnelle,et cela,pour les lecteurs qui ne sont pas habitués à notre démarche.Ce chapitre doit donc être considéré comme une introduction au sujet. Pour plus de détail,on se réfèera à [A3].

L'analyse fonctionnelle d'un projet informatique a pour but de décrire rigoureusement deux choses :

- les traitements qui seront touchés par l'informatisation
- les données qui seront nécessaires pour exécuter ces traitements.

Dans le cycle de vie d'un projet informatique,l'analyse fonctionnelle suit directement l'analyse de conception d'un système informatique.Rappelons brièvement ce que l'on entend par analyse de conception[A2].

Lors de cette étape,le concepteur établira les plans généraux du projet d'informatisation de son système d'informations.

Cette analyse se décompose en plusieurs points.
Dans un premier temps,l'analyste devra identifier les objectifs de l'organisation.Il est important de déterminer les raisons pour lesquelles on désire éventuellement informatiser certains traitements.

De ces objectifs,on déduira les objectifs informatiques du futur système. Pour le cas Petitpas,ces divers objectifs sont exposés lors de la description générale de la firme,et cela,au chapitre trois du mémoire.

L'identification des objectifs du projet terminée,le responsable devra découvrir les ressources et les contraintes du système:

- contraintes matérielles, financières, organisationnelles, humaines ou institutionnelles (lois)

- ressources matérielles,humaines,financières.

Parallèlement,cette personne prendra connaissance de l'existant : détermination des traitements,des données en input et en output,des ressources utilisées, ce qui permet de mettre en évidence les points critiques du problème à résoudre.

L'étape suivante de la démarche consistera en la structuration de différentes solutions.Pour notre problème,il serait également possible d'envisager une solution manuelle ou de mettre en place une solution utilisant un réseau, avec donc un système distribué et non centralisé comme nous,nous la réaliserons.

De ces diverses solutions,il faudra alors en retenir une.Les critères permettant ce choix sont multiples,citons comme exemples :le critère économique (la solution la plus rentable),le critère organisationnel(le moins de changement dans la firme),le critère informatique(délai de réponse,fiabilité,simplicité), le critère social(le moins de licenciements possible).

Ce choix conduira à l'écriture d'un dossier reprenant tous les points précédents liés à cette solution.

Nous arrivons,ainsi,au deuxième stade du cycle de vie d'un projet informatique : l'analyse fonctionnelle.Elle consiste à prendre la solution choisie et à la décrire de façon rigoureuse et précise.

Cette analyse débouche sur un dossier,en ce qui nous concerne le document ISDOS(que nous expliquerons ci-dessous),qui traite des deux points principaux : la description des données, la description des traitements.

Pour la description des informations,l'analyste constitue un dictionnaire des données,qui reprend,sous une forme ou une autre,la liste complète des données que le système informatique traitera.Il est important que ce dictionnaire existe,car toutes ces données doivent avoir un sens précis connu de tous.

L'analyste établit,également une structuration des informations.En effet, ces informations ne se trouvent pas toutes sur le même pied;par exemple,la notion générale de client n'est pas au même niveau que la notion de code postal de la localité où habite le client.

Il existe donc des données élémentaires (le code postal) et des données complexes(un client)qui comprennent des données élémentaires(nom,prénom,code postal ...).

Pour être complet,on joindra à ce rapport une quantification concernant les informations structurées(le nombre de client de la firme).

Pour la description des traitements,l'analyste étudie ces derniers de deux façons : de manière statique,puis de manière dynamique.

Dans la partie statique,l'informaticien décrit ce que réalise le processus(son objectif),les données sur lesquelles il travaille,les messages qu'il reçoit ou qu'il génère.

Dans la partie dynamique du système,l'informaticien montre la façon dont les traitements s'enchaînent,les points de synchronisation et les évènements attendus.

L'analyse fonctionnelle peut permettre d'établir une maquette d'évaluation du projet.

Méthode de l'analyse fonctionnelle.

La méthode ISDOS employée pour présenter l'analyse fonctionnelle du cas Petitpas,est enseignée à l'Institut d'informatique de Namur.Cette méthode développée aux Etats-Unis et reprise par l'unité de gestion,se base sur le modèle Entité-Association.Nous n'expliquerons pas ici ce modèle,mais en parcourant cette présentation succinte d'ISDOS,on peut en comprendre les principales notions.Pour plus de détail,on se réfèrera à [A4] et à [A6].

Nous allons donc,maintenant,présenter le langage contenu dans ISDOS[A5], pour mieux saisir cet exposé,le lecteur devrait lire en parallèle le document reprenant l'analyse fonctionnelle complète de Petitpas.

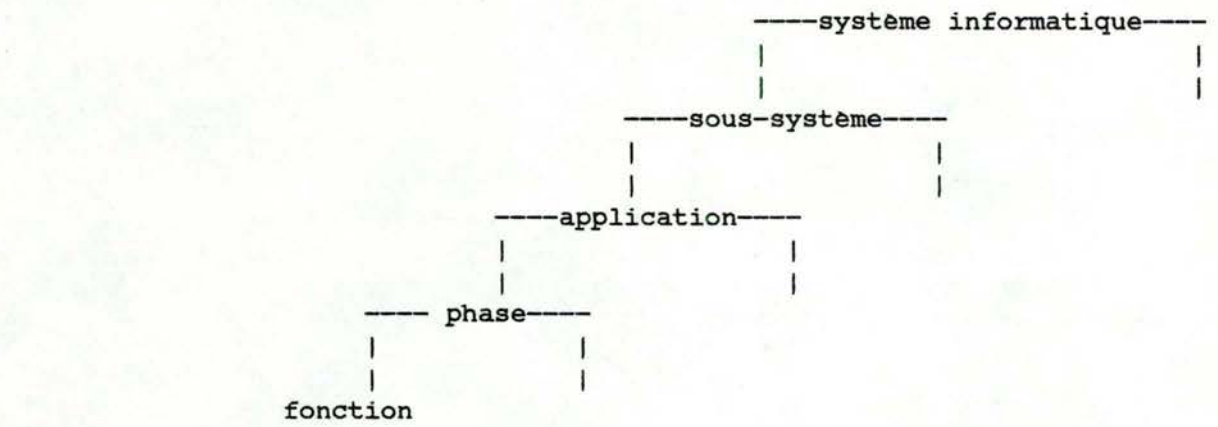
Le document ISDOS est divisé en sept parties :

- la structure du système
- le modèle conceptuel des données
- le dictionnaire des données
- le flux du système
- la dynamique du système
- les interfaces
- les quantificateurs.

La structure du système.

Dans cette partie,on décrit la structure des traitements de manière globale.

Un système informatique peut se décomposer en plusieurs niveaux :



Voyons ce que cela donne pour le cas Petitpas.Un sous-système de la firme est l'application de gestion des clients et des fournisseurs,un autre pourrait être la comptabilité de l'entreprise.Nous avons pour ce sous-système l'application client et l'application fournisseur,la première contient la phase enregistrement et contrôle de la commande client dont une fonction pourrait être préparation du bon de commande client.

Il existe des règles qui définissent de manière précise une phase et une fonction,mais nous n'en parlerons pas dans le cadre de ce mémoire(se référer à [A3]).

Dans le langage employé par ISDOS,on ne retrouve pas ces différentes notions(fonction,phase,application....). Mais il existe le vocable PROCESS et par le vocable ATTRIBUTES d'un PROCESS, on peut définir le type de PROCESS dans le hiérarchie définie.

Le modèle conceptuel des données.

Dans cette partie,on décrit les données complexes de l'analyse fonctionnelle.Deux notions importantes sont reprises : ENTITY et RELATION. Ce sont les bases du modèle Entité-Association.Une entité est ce qu'un individu voit comme un tout dans le système d'informations,par exemple le client,le fournisseur ,la commande du client.Elle est composée (CONSIST OF) d'un ensemble d'éléments et elle peut être identifiée par un de ces éléments,par exemple le numéro du client pour l'entité client.

Une relation relie plusieurs entitès,par exemple la ligne de commande client qui lie commande client et produit.A ces relations,sont parfois associées des données(ASSOCIATED DATA).

On remarque que ces deux notions sont souvent regroupées en SET(ensemble d'informations) qui rèunit les informations utilisables par une phase.

Le dictionnaire des données.

Cette partie dècrit l'ensemble des données èlèmentaires du sous-système. Trois notions sont prèsentes : l'èlèment(ELEMENT) qui est la donnée la plus fine (le nom du client),le groupe (GROUP) qui regroupe des èlèments (l'adresse du client qui peut se dècomposer en rue,numèro,localitè, code postal),le message (MESSAGE) qui est transmis entre les composants du système ou entre le système et son environnement (une fiche de rèapprovisionnement). Pour ces trois types d'information, on donne surtout une description expliquant en français la signification de la donnée.

Le flux du système.

Cette partie dècrit l'ensemble des fonctions du cas Petitpas.Pour chaque PROCESS,on donne une description de la fonction (objectif,règles) ,sa classe (manuelle ou automatique),les messages qu'elle reçoit(RECEIVES) et qu'elle gènère(GENERATES) sous certaines conditions (IF),et la structure de données qu'elle utilise(USES).

La dynamique du système.

Cette partie dècrit l'enchaînement des fonctions.Pour une fonction on y indique le fait qui lance son exècution(TRIGGERED), ce fait est souvent la terminaison d'un autre processus.Ce chapitre comprend ègalement,les notions de point de synchronisation entre fonctions et d'èvènement (l'arrivèe de marchandises).

Les interfaces.

Cette partie dècrit les interfaces du système,par exemple le client physiquement parlant(à ne pas confondre avec l'entitè

client,interne au système).

Les quantificateurs.

Cette partie donne quelques quantificateurs utilisés dans les chapitres précédents.

4.2 Analyse fonctionnelle du cas Petitpas.

L'analyse fonctionnelle du cas Petitpas a été réalisée pour les travaux pratiques qui accompagnent le cours théorique d'analyse fonctionnelle donné en première licence par Mr.Bodart.

Notons au passage qu'il ne s'agit pas de la dernière version de l'analyse de Petitpas.D'ailleurs,cette étude du système informatique de la firme continue de s'affiner d'année en année.Il nous fallait donc nous arrêter sur une version.Celle qui fut choisie,de l'année 1978-1979,est très proche du dernier exemplaire.

Remarquons également qu'il n'était pas de notre ressort dans le cadre du mémoire,de critiquer cette version ni de l'améliorer,nous avons donc gardé ses qualités et ses défauts.

Dès le départ de notre travail,il fut décidé de nous limiter à une partie seulement de l'application Petitpas,ce sous-ensemble devant couvrir toutes les caractéristiques informatiques de la totalité du sujet.

Quel est ce sous-ensemble ?

Les deux applications,traitement des commandes clients et traitement des livraisons fournisseurs,ont été abordées.En ce qui concerne les commandes clients,nous nous sommes attachés à réaliser les fonctions suivantes :

- pour la phase "enregistrement et contrôle de la commande client":

  + "préparation du bon de commande client" (manuelle)

  + "enregistrement et contrôle du bon de commande client"

  + "traitement des cas litigieux" (manuelle)

- pour la phase "mise à jour moins du stock":

  + "mise à jour moins suite à l'enregistrement d'une commande client"

  + "mise à jour moins suite à une sélection différée"

  + "mise à jour moins pour un produit épuisé"

  + "mise à jour moins du stock".

Ont été laissées de coté,les phases suivantes :

+ "préparation de la commande client" (en partie manuelle)
  (ordonnancement)

+ "emballage et facturation" (en partie manuelle).

Pour les livraisons des fournisseurs,nous avons traité les fonctions suivantes :

- pour la phase "réception d'une livraison fournisseur":

  + "contrôle et emmagasinage" (manuelle)

  + "enregistrement de la livraison fournisseur"

- la totalité de la phase "mise à jour plus du stock"

par contre,n'a pas été traitée la phase suivante :

- "préparation et enregistrement de la commande fournisseur".

Les entités et les relations nécessaires pour exécuter ces traitements ont été reprises,avec la totalité des éléments qu'elles comprennent.

Pourquoi ce choix ?

Comme nous l'avons dit plus haut,le sous-ensemble réalisé se doit de posséder tous les problèmes informatiques de l'application.

Mais c'est au niveau de l'architecture que les problèmes ont été détectés,c'est donc dans le chapitre suivant que nous expliquerons la raison de ce choix.

CHAPITRE 5 : ANALYSE ORGANIQUE.

Jean-Pierre Bodart

5. Analyse organique [A13][A9][A11].

Introduction.

Dans la première partie de ce chapitre [5.1], nous allons présenter d'une manière générale ce que recouvre l'analyse organique enseignée à l'Institut d'Informatique; ensuite dans la seconde partie [5.2], nous allons présenter l'analyse organique du cas Petitpas en respectant la démarche énoncée dans la première partie.

5.1. Contexte, but et méthode de l'analyse organique.

5.1.1. Contexte et but de l'analyse organique.

La démarche d'analyse développée à l'Institut d'Informatique est constituée de plusieurs étapes. Les deux premières sont l'analyse de conception et l'analyse fonctionnelle qui ont été présentée dans le chapitre 4.

Ensuite, vient l'étape qui nous intéresse, l'analyse organique encore appelée analyse d'implémentation qui propose des moyens pour réaliser la solution fonctionnelle résultat du degré précédent de notre démarche. A ce stade , un certain nombre de contraintes sont prises en compte quant à l'identification des fichiers et des programmes. Le résultat de l'analyse organique est appelé solution implémentable et constitue la donnée d'entrée de la phase suivante qu'est l'implémentation. Celle-ci consiste à adapter et réaliser la solution implémentable de telle façcon qu'elle fonctionne sur la machine (ce terme regroupant à la fois les outils "hardware" et logiciels de base) choisie par ailleurs. Nous disposons alors d'une solution implémentée, laquelle subira

---

[A13] Concepts, méthodes et outils de l'analyse organique
(Notes de cours personnelles).
A. van Lamsweerde 1979-1980

[A9] Fichiers et banques de données(deuxième partie)
(Notes de cours personnelles).
J-L. Hainaut 1979-1980

[A11] Analyse du cas "Petitpas"
Dossier d'analyse organique
Première partie : la base de données
J-L. Hainaut 1979-1980

une phase de tests de la part des utilisateurs au terme de laquelle la solution sera dite exécutable et exploitable. Enfin l'étape ultime consiste en la maintenance de la solution exécutable; les raisons de cette phase sont d'ordres divers :

- changements fonctionnels,
- erreur de logiciel de base,
- programmation,
- évolution de la législation, etc ...

Cette maintenance implique que ce cycle de vie est itératif. Il est à remarquer que ces diverses étapes qui forment le cycle de vie d'un projet informatique n'ont pas de frontières rigides entre elles.

Maintenant que nous avons situé l'analyse organique dans son contexte, explicitons la.

5.1.2. Méthode de l'analyse organique.

L'analyse organique est elle-même constituée de plusieurs étapes :

- détermination d'une structure d'accès aux données,
- établissement d'une architecture,
- développement de modules de traitement et d'accès aux données,
- validation,
- évaluation des performances.

a) Détermination d'une structure d'accès aux données [cfr A9].

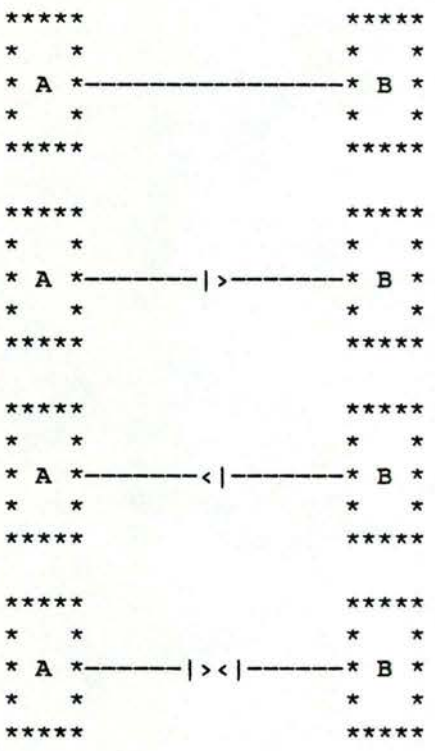
Introduction.

La détermination de la structure d'accès procède d'une démarche en deux étapes. La première consiste en l'élaboration du schéma conceptuel binaire, ensuite vient l'étape de la construction du schéma des accès possibles.

A ce schéma sont associées des contraintes de quatre types, à savoir :

- les contraintes d'existence,
- les contraintes liées à la force des types de chemin; on définit un type de chemin comme étant fort pour un type d'article origine ou cible si tout article de ce type doit être à tout instant l'origine ou la cible d'au moins un chemin de ce type.
- les contraintes de valeur,
- les contraintes de connectivité des types de chemin.

Le formalisme utilisé pour ces dernières est le suivant :



Une relation "x - y" signifie qu'à une occurrence de A correspondent au plus y occurrences de B et inversement à une occurrence de B correspondent x occurrences de A.

Schéma conceptuel binaire.

Le schéma conceptuel dont le formalisme est celui d'un modèle binaire, est élaboré à partir du schéma fonctionnel déterminé lors de l'analyse fonctionnelle; celui-ci étant représenté dans un modèle entité-association, une transformation s'impose. Les principes de cette dernière figurent dans l'annexe 1.

Schéma des accès possibles.

L'élaboration du schéma des accès possibles se fait de la manière suivante à partir du schéma conceptuel binaire :

- à tout type d'entité est associé un type d'article, chaque type d'entité étant représenté par un article,
- à tout attribut est associé un item,
- à toute relation sémantique sont associées deux relations d'accès, qui lui sont équivalentes, inverses l'une de l'autre,
- un accès séquentiel est prévu pour tout type d'article.

Un tel schéma ignore les critères de performance, de sécurité, de concurrence, ...

b) Etablissement d'une architecture.

Comme déjà précisé, notre but principal n'était pas de réaliser toute l'application Petitpas, mais de réaliser un sous-système représentatif permettant de mettre en évidence tous les problèmes suscités par l'application.
Pour Petitpas, les problèmes nous semblent être les suivants :

L'établissement d'une architecture consiste d'une part à réaliser à partir de la solution fonctionnelle, des décompositions et/ou des regroupements de fonctions en programmes, lesquels incluent des algorithmes (= traduction des règles venant de l'analyse fonctionnelle), des opérations d'accès aux données et des algorithmes système, et d'autre part, à constituer une structure de programmes.

Identification des modules de programme.

Définition d'un module de programme [cfr A13].

C'est un composant d'un système qui satisfait à certains critères. Ces critères de décomposition en modules sont nombreux, c'est pourquoi nous ne citerons que ceux qui nous semblent les plus importants :

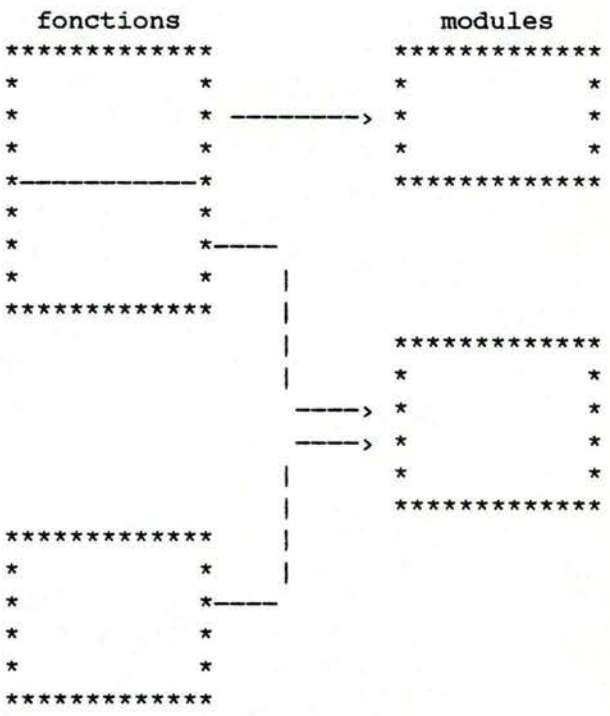
- la dimension d'un module,
- le degré de couplage entre modules exprimé par le nombre et la complexité des connexions entre modules,
- la souplesse d'intégration d'un module, traduisant la possibilité d'imbrications faciles entre modules; une condition permettant de respecter ce but est de définir un seul point d'entrée, de même qu'un seul point de sortie par module,
- la cohésion interne d'un module traduite par le degré d'interdépendance entre les actions et les conditions d'un module,
- la minimisation du nombre de consultations d'une même information,
- le maintien du parallélisme maximum de déroulement des différents traitements.

Modularisation.

La décomposition de notre application en modules respecte la décomposition fonctionnelle. La correspondance entre modules et fonctions peut prendre deux formes :

- groupage de plusieurs fonctions dans un même module,
- éclatement d'une même fonction en plusieurs modules.

Cependant pour respecter l'acquis de la découpe rationnelle en fonctions, le cas de figure suivant ne peut se présenter :



L'architecture globale des traitements procède en général d'une intégration de modules d'"entrée-sortie" et système aux modules fonctionnels.

c) Développement de modules de traitement et d'accès aux données.

La phase de développement de modules de traitement et d'accès aux données intervient une fois que l'architecture est figée. Son but est d'abord de spécifier un module, c'est-à-dire préciser ce que le module doit réaliser. Ensuite, vient l'étape de conception et représentation du module, au cours de laquelle seront réalisés l'algorithme répondant à la spécification, le choix de la représentation des structures de données et la détermination des opérations d'accès aux données.

Spécification des modules de traitement.

La spécification des modules consiste à préciser ce que doit réaliser un module et sur quelles données. Pour une plus grande généralité, la spécification doit ne pas être algorithmique. Nous proposons ci-après la méthode dite des "pré- et post-conditions" qui satisfait à cette exigence.

Formalisme.

Les quatre éléments qui concourent à la spécification sont :

- l'argument d'entrée : il précise les données que le module reçoit en entrée,
- l'argument résultat : il précise le résultat engendré par l'exécution du module,
- l'argument pré-condition : il précise les conditions exigées pour que l'exécution du module puisse démarrer,
- l'argument post-condition : précise l'état terminal consécutif à l'exécution du module.

Conception et représentation des modules.

Formalisme[cfr A13].

Nous avons utilisé un pseudo-langage de description structurée d'algorithmes du type Algol ou Pascal utilisant un vocabulaire restreint et une syntaxe simplifiée.
Ce pseudo-langage possède les caractéristiques suivantes :

- des structures de contrôle logique :
  - séquence d'instructions(verbes à l'infinitif et opérations arithmétiques élémentaires),
  - décision (if ... then ... else ...),
  - itération (while B do S od),
- des opérations d'accès aux éléments d'une structure de données. Les accès identifiés sont les suivants :
  - accès simple :

```
for identificateur = donnée_cible
from donnée_origine
do T od
```

  - accès itératif :

```
for each identificateur = donnée_cible
from donnée_origine
do T od
```

  - accès sélectif :

```
for each identificateur = donnée
such that condition
do T od
```

- des opérations de modification et de recherche d'éléments de la structure : créer article, supprimer article, mettre-à-jour article, rechercher article, ...

- des opérations d'entrée-sortie :

  - en entrée : for introduit do S od,
  - en sortie : afficher article ou message,

- la qualification des données : propriété(:article),

- des appels à des routines : appel nom_de_routine,

- la communication de données entre programmes :

  - envoyer donnée à programme,
  - recevoir donnée de programme.

d) Validation.

La validation du produit logiciel consiste à vérifier la fiabilité du projet. Trois approches permettent d'atteindre ce but :

- tests,
- vérification,
- dérivation de programmes corrects par construction.

L'approche par test inclut elle-même deux voies : la première consistant en une inspection systématique du programme selon une liste d'erreurs les plus fréquentes (déterminées par les statistiques) et la seconde reposant sur la conception de jeux de test ayant une grande probabilité de découvrir des erreurs. le principe de cette seconde méthode est basé sur la définition d'un critère et la construction d'un ensemble de jeux de test satisfaisant à ce critère.

Voici deux types de test répondant à ce dernier principe :

- le test dit "black box" consiste à couvrir :

  - chaque classe d'équivalence de données identifiée dans les spécifications,
  - chaque cas limite identifié dans les spécifications,
  - chaque relation de cause à effet (par ex. : table de décision séquentielle),

- le test dit "white box" consiste à couvrir :

  - chaque opération,
  - chaque condition élémentaire d'une décision,
  - chaque décision,
  - toutes les combinaisons possibles de valeur de condition,
  - un ensemble représentatif de chemins d'exécution.

  Ce type de test oblige à connaître l'algorithme du module.

A ces types de tests s'ajoutent des tests permettant la validation d'une architecture de programmes. Ce sont les tests d'intégration :

- test incrémental consiste à d'abord tester le module individuellement , ensuite tester sa combinaison à tous les modules déjà testés.

- test non incrémental comporte deux stades :

  - test individuel de chaque module,
  - test d'un ensemble pertinent de combinaisons de modules.

  Une explication plus détaillée de ces tests sera trouvée dans [A13].

L'approche par vérification réside dans la démonstration mathématique de la cohérence de l'algorithme avec les spécifications d'entrée et de sortie.

L'approche par dérivation de programme repose sur la construction du programme simultanément à sa démonstration de validité.

Ces deux dernières démarches n'ont pas été approfondies, mais mentionnées à titre d'indication.

e) Evaluation des performances.

A ce niveau, on peut effectuer une première évaluation des performances en coût d'accès à la base de données.

5.2. Analyse organique de Petitpas.

Conformément à ce qui fut décidé au début de notre travail et mentionné au chapitre 4, l'analyse organique de l'application Petitpas se limite aux fonctions citées au chapitre précédent.

5.2.1. Structure d'accès de Petitpas.

a) Schéma conceptuel binaire et contraintes de Petitpas.

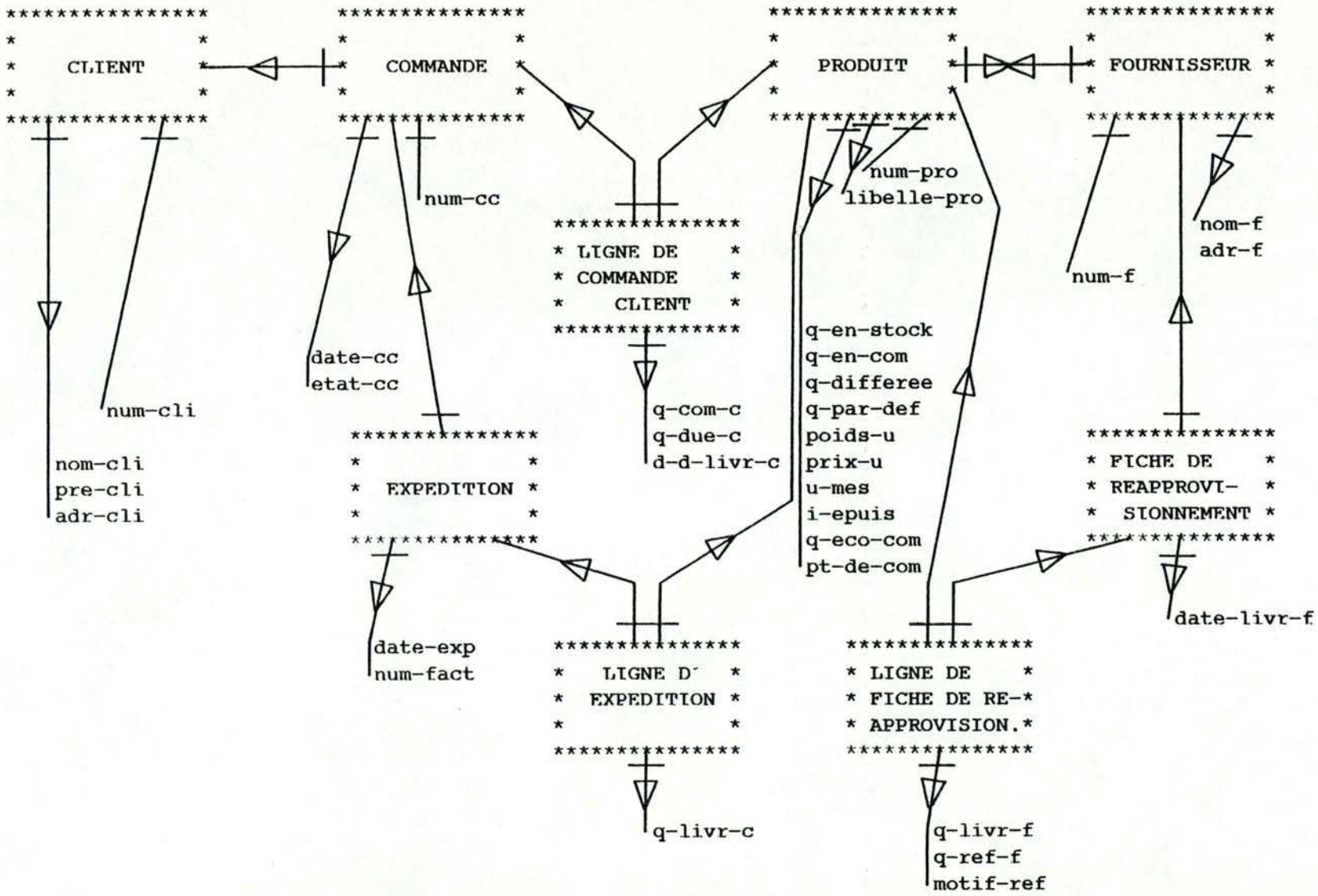


Schéma conceptuel binaire de Petitpas.

Contraintes de Petitpas.

a) Contraintes de cardinalité.

Ces contraites expriment la cardinalité existant entre les types d'article qui composent une relation.Dans la suite,n représente un nombre entier theoriquement égal à l'infini(en realite ce nombre sera limité et égal à une valeur découlant de l'étude de l'existant). Un couple de nombres comme par exemple 0-n indique une valeur minimum et une valeur maximum.

- relation CLIENT<-->COMMANDE-CLIENT
  à un CLIENT correspond 0-n COMMANDE-CLIENT
  à une COMMANDE-CLIENT correspond 1-1 CLIENT

- relation COMMANDE-CLIENT<-->LIGNE-CMDE-CLIENT
  à une COMMANDE correspond 1-n LIGNE-CMDE-CLIENT
  à une LIGNE-CMDE-CLIENT correspond 1-1 CLIENT

- relation LIGNE-CMDE-CLIENT<-->PRODUIT
  à une LIGNE-CMDE-CLIENT correspond 1-1 PRODUIT
  à un PRODUIT correspond 0-n LIGNE-CMDE-CLIENT

- relation COMMANDE-CLIENT<-->EXPEDITION
  à une COMMANDE-CLIENT correspond 0-n EXPEDITION
  à une EXPEDITION correspond 1-1 COMMANDE-CLIENT

- relation EXPEDITION<-->LIGNE-D-EXPEDITION
  à une EXPEDITION correspond 1-n LIGNE-D-EXPEDITION
  à une LIGNE-D-EXPEDITION correspond 1-1 EXPEDITION

- relation LIGNE-D-EXPEDITION<-->PRODUIT
  à une LIGNE-D-EXPEDITION correspond 1-1 PRODUIT
  à un PRODUIT correspond 0-n LIGNE-D-EXPEDITION

b) Contraintes d'existence.

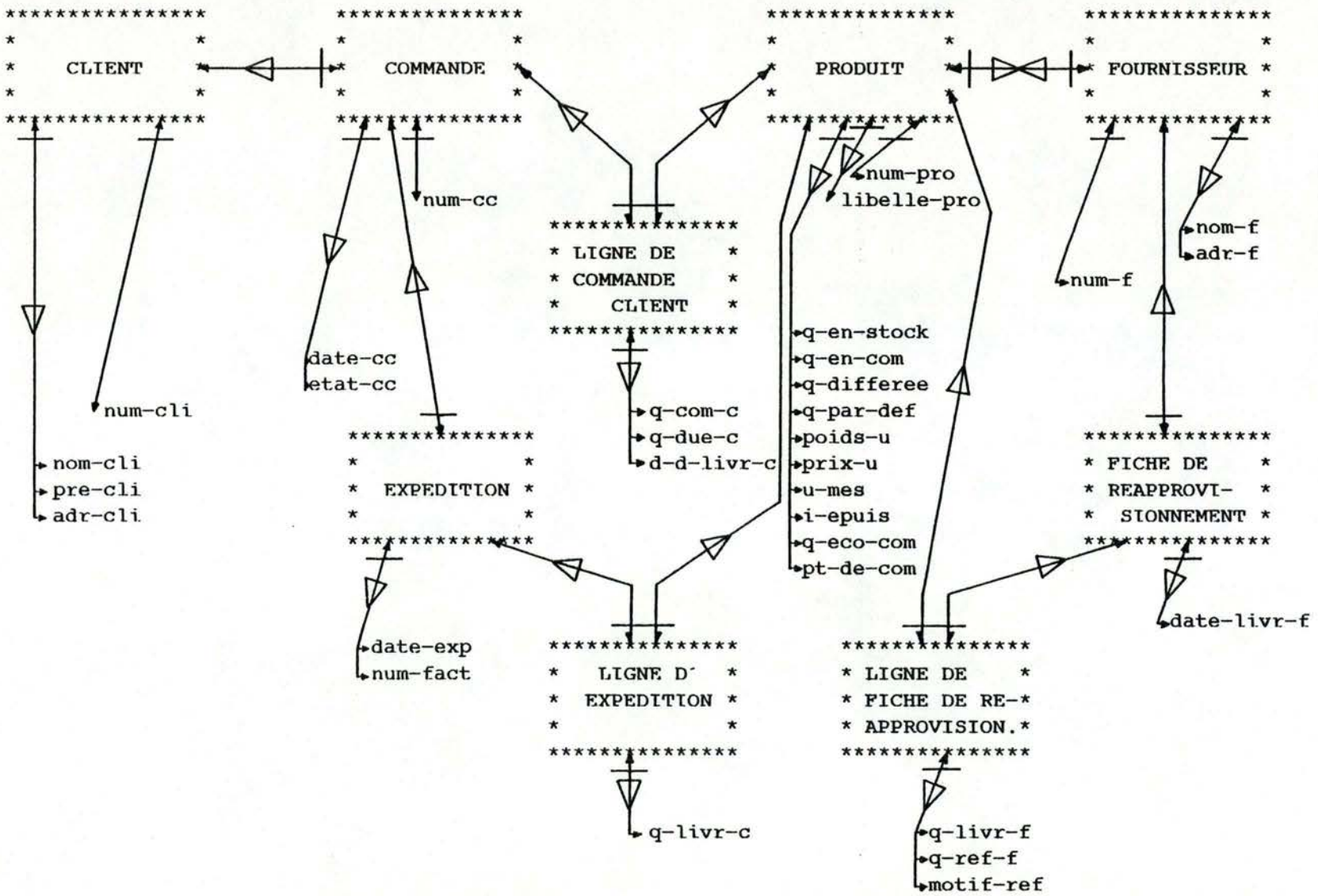
Des contraintes de cardinalité ,on déduit:

COMMANDE-CLIENT existe si CLIENT existe

LIGNE-CMDE-CLIENT existe si COMMANDE-CLIENT et PRODUIT existent

EXPEDITION existe si COMMANDE-CLIENT existe

LIGNE-D-EXPEDITION existe si EXPEDITION et PRODUIT existent



b) Schéma des accès possibles de Petitpas.

5.2.2. Etablissement d'une architecture de programmes.

Comme déjà précisé, notre but n'était pas de réaliser toute l'application Petitpas, mais de réaliser un sous-système représentatif permettant de mettre en évidence tous les problèmes suscités par l'application.

L'application Petitpas nous semble comprendre les problèmes suivants :

- les traitements accèderont aux fichiers de manière directe et séquentielle, par des chemins identifiants et non identifiants,par des clés numériques et alphanumériques,
- il faudra prévoir une protection des données et une synchronisation entre les accès concurrents,
- nous aurons des communications de messages entre des processus internes au système ou entre le système et le monde extérieur(dialogue avec les utilisateurs). Cette communication devra se faire entre un nombre quelconque de processus. De plus,des priorités entre messages peuvent exister.
- certaines tâches existeront en plusieurs exemplaires dans le système.

On notera que ces problèmes peuvent dépendre de l'architecture choisie.

Le sous-ensemble déterminé au point 4.2 reprend-t-il toutes ces difficultés ?

- nous aborderons effectivement le problème d'accès aux fichiers,de protection d'accès et de synchronisation,

- nous aborderons également le problème de communication entre processus:

    - avec dialogues ("enregistrement et contrôle de la commande client")
    - avec priorité de messages ("mise à jour moins suite à une commande différée ou enregistrée")
    - avec des relations entre plusieurs processus(plusieurs processus communiquent avec "mise à jour moins ")

- certains processus existent en version unique d'autres en version multiple ("enregistrement d'une livraison fournisseur" existe en version multiple).

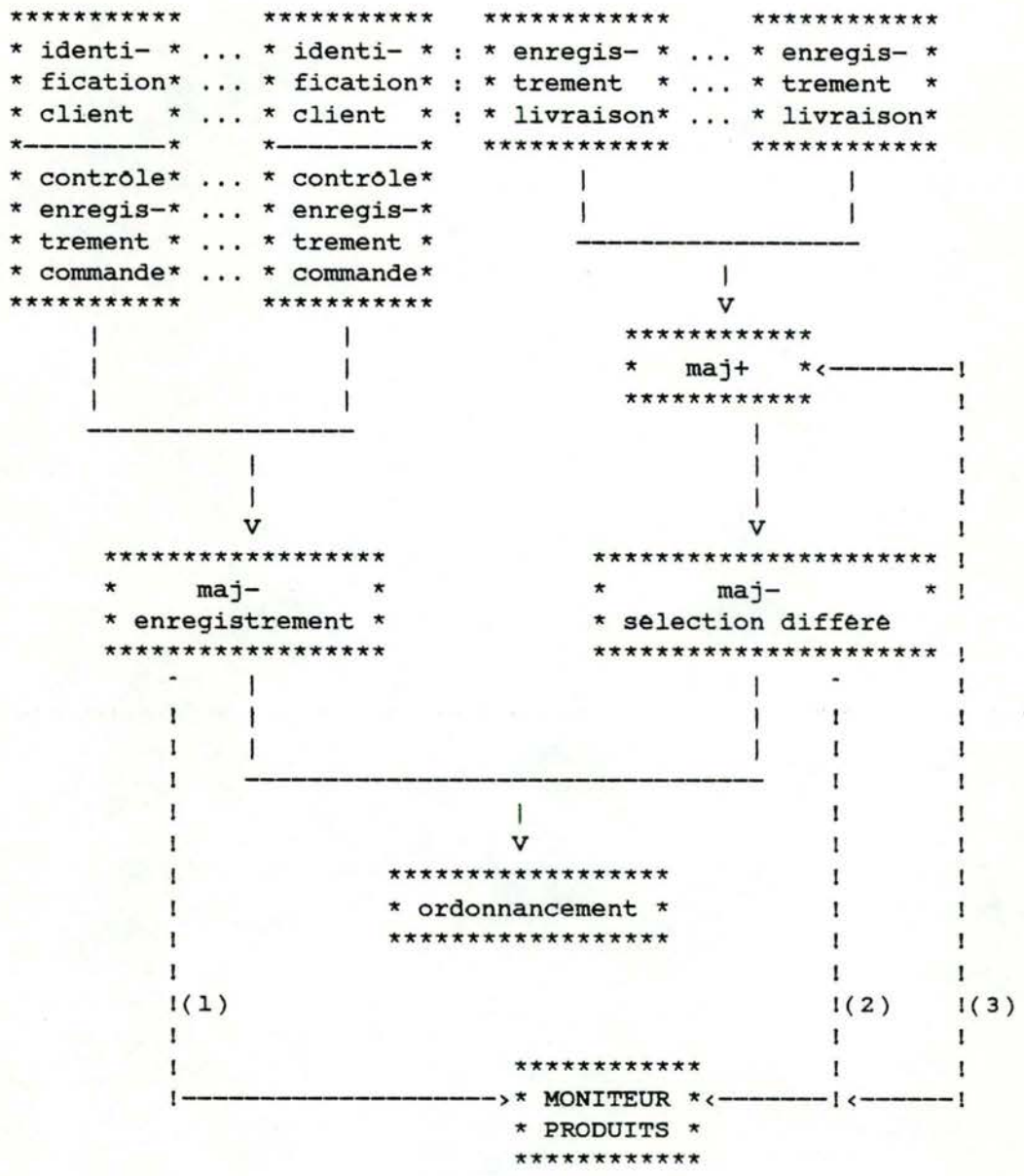
Nous retrouvons donc les caractéristiques informatiques du cas général.

Nous proposons ci-après l'architecture qui nous a été proposée au cours de nos études. Ensuite nous discutons ce choix et présentons notre solution.

a) Schéma de la solution de l'Institut.

Remarque.

Nous l'avons restreinte au sous-ensemble que nous devons réaliser.



Les flèches représentent le sens de la transmission de données ou d'envoi de signaux.

Les communications (1),(2) et (3) représentent des signaux, tandis que les autres représentent des messages.

"..." séparant deux rectangles signifie que le nombre de copies des modules de programme que représentent ces rectangles est quelconque,

":" séparant deux rectangles met en évidence le fait que les modules de programme représentés par ces derniers n'appartiennent pas au même flux de données. Rappelons que dans le sous-ensemble de Petitpas que nous avons réalisé, deux flux de données apparaissent : celui des commandes clients et celui des

livraisons fournisseurs,

Le ou les séparations à l'intérieur d'un rectangle illustre le fait que le module de programme est constitué de plusieurs fonctions mises en évidence lors de l'analyse fonctionnelle.

Critique.

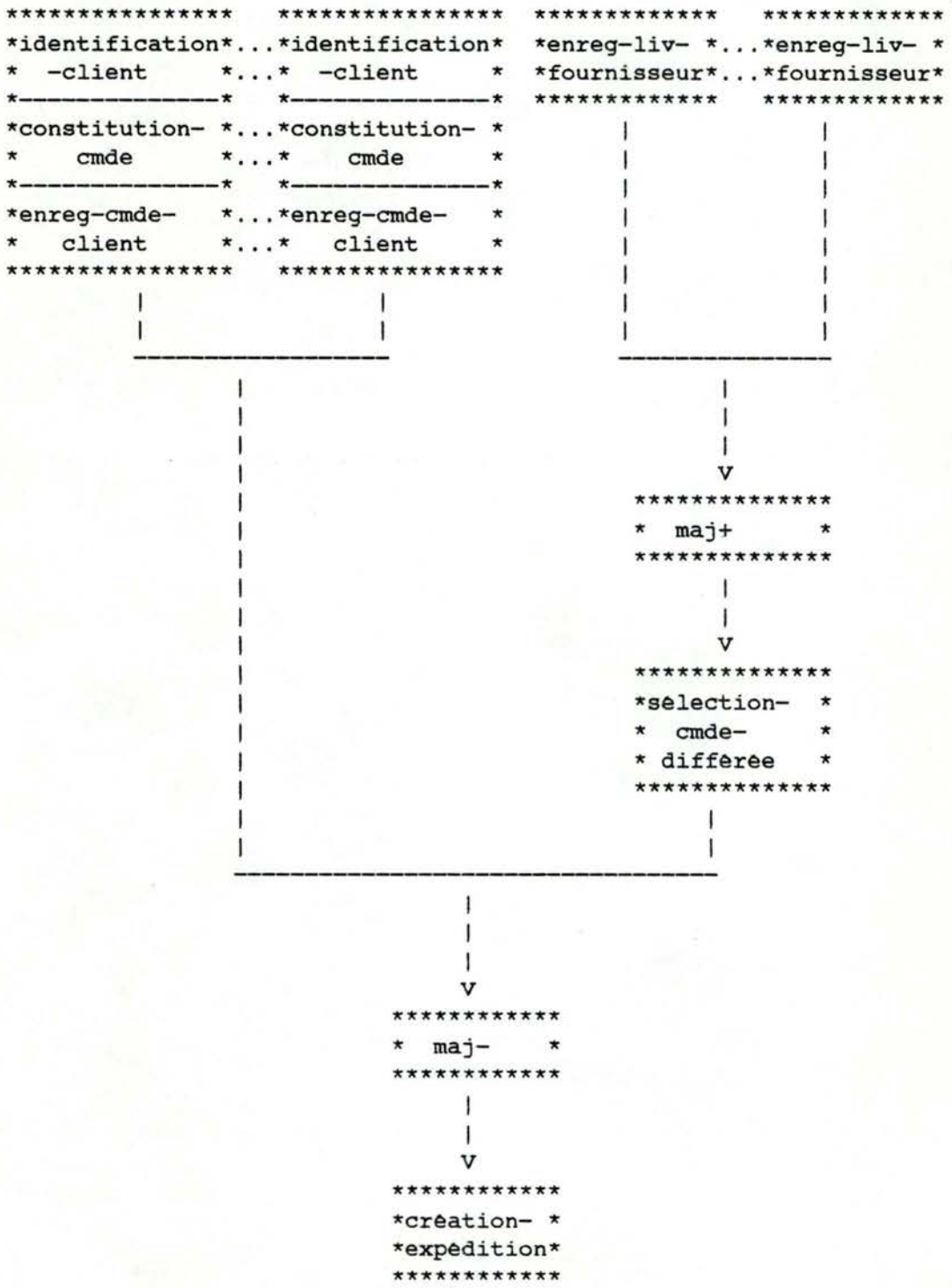
Dans cette architecture, on remarque deux types de module d'une part les modules fonctionnels et d'autre part un module système appelé "moniteur produit" qui règle les conflits de priorité fonctionnelle entre la "maj-" suite à une commande différée, la "maj-" suite à une commande du jour et la "maj+". Le "moniteur produit" reçoit des signaux de la part des modules fonctionnels cités ci-avant traduisant une demande d'exécution et envoit des signaux de permission d'exécution; le nombre de communications de synchronisation est donc important.

Quant au degré de parallélisme, nous constatons qu'il est maximum; en effet, à chaque fonction correspond un programme sauf en ce qui concerne l'"identification du client" et le "contrôle et enregistrement d'une commande client" qui ont été regroupées en un programme. Ces deux fonctions étant de type interactif, l'efficacité de l'architecture n'aurait pas été plus grande si l'on en avait fait deux programmes, car l'identification d'un nouveau client ne peut se faire que si le contrôle et l'enregistrement de la commande du client précédent est terminée. De plus, il y aurait eu une communication supplémentaire entre programmes.

Une solution inacceptable aurait été d'associer un programme unique à la fonction de "contrôle et enregistrement d'une commande client", car à ce moment ce programme devait être partagé entre plusieurs utilisateurs, or comme ce programme est interactif son temps de réponse est à l'échelle humaine et non à l'échelle machine; ainsi, le temps de réponse serait devenu inacceptable. Ceci prouve donc que la solution est optimale quant au degré de parallélisme.

b) Solution proposée.

Schéma de la solution.



Critique.

Nous proposons une solution autre cependant fortement inspirée de la précédente, mais où nous fusionnons trois modules : celui de la "maj-" suite à l'enregistrement d'une commande du jour, celui de la "maj-" suite à la sélection d'une commande différée et le "moniteur produit".
Cette solution garde le degré de parallélisme de la première, tandis qu'elle diminue le nombre de communications entre programmes et le nombre de programmes dans le système.

Quant aux priorités de la mise à jour suite à une commande différée sur la mise à suite à une commande du jour mise en exergue lors de l'analyse fonctionnelle, nous avons estimé qu'un respect strict de ces priorités n'avait aucun sens au niveau des programmes. La décision de respecter ces priorités doit à notre sens être prise au niveau organisationnel et son processus complètement défini. Ainsi par exemple, si l'on veut respecter strictement la priorité d'une mise à jour moins suite à une sélection des commandes différées sur la mise à jour moins suite à des commandes du jour, aucune commande du jour ne peut être traitée tant que l'enregistrement des livraisons fournisseurs n'est pas terminé. Si le réapprovisionnement est très court, le respect des priorités est envisageable, mais à notre avis, on ne peut garantir un temps de réapprovisionnement ponctuel; c'est la raison qui nous fait rejeter le strict respect des priorités fonctionnelles. Nous estimons que l'on ne gagnerait rien à respecter cela.

Une analyse plus approfondie des fonctions de mise à jour moins suite à l'enregistrement des commandes du jour et mise à jour moins suite à une sélection des commandes différées, nous a permis de remarquer que le traitement réalisé est pratiquement identique pour une commande client du jour et une commande client différée. Nous avons donc décidé de fusionner ces deux fonctions en un seul module de programme, dans lequel nous avons intégré la gestion des priorités réalisée par le "moniteur produit". Ceci permet de diminuer le nombre de modules et le nombre de communications.

Nous avons décidé de ne pas réaliser l'ordonnancement, car d'une part les spécifications sont incomplètes (par exemple pour le parcours en magasin, il faut connaître la disposition des rayons à produits à l'intérieur de celui-ci, etc ...), et d'autre part la réalisation de cette fonction n'apporte pas de problèmes nouveaux de réalisation de l'architecture de Petitpas et d'autre part nous avons préféré réaliser un un module d'expédition simple (en ce sens que sa fonction consiste à lister le fichier des expéditions) nous permettant ainsi de matérialiser le fonctionnement de ce que nous avons réalisé.

5.2.3. Développement des modules de Petitpas.

Conventions.

"/\" représente le "et" logique,
"\/" représente le "ou" inclusif logique,
"∈" représente l'appartenance de l'algèbre ensembliste,
"∉" représente la non-appartenance de l'algèbre ensembliste,
"∀" représente le signe "pourtout" de l'algèbre ensembliste.

Spécification des modules de Petitpas.

Voici un exemple de module de l'application Petitpas spécifié selon la méthode présentée plus haut; les autres modules sont spécifiés à l'annexe 1.

module <idclient>

Argument :

client : bon_commande_accepté.

Résultat :

numéro-client pour <enreg_cmde>.

Précondition :

néant.

Postcondition :

(client : bon_commande_accepté appartient fichier client) " \/" ((client : bon_commande_accepté n'appartient pas fichier client) "/\" (fichier client := fichier client + client : bon_commande_accepté)).

module <enreg_cmde>.

Argument :

numéro_client de <idclient>.

Résultat :

numéro_commande_client pour <majmoins>
ou bon_commande_refusé pour <traitement_cas_litigieux>.

Précondition :

numéro_client appartient fichier client.

Postcondition :

```
(( commande acceptée) "/\" (fichier commande_client :=
fichier commande_client + bon_commande_accepté) "/\"
(numéro_commande pour <majmoins>))
 "\/" ((commande incorrecte) "/\" (bon_commande_refusé pour
<traitement_cas_ litigieux>)).
```

Conception et représentation des modules de Petitpas.

Nous donnons un exemple d'application de la méthode précisée plus haut.

<idclient>

```
do
     recevoir
      ci : client(nom,prénom,adresse,numéro) from terminal;
     chercher_client (arg : numéro : ci);
     if not trouvé
     then
          do
          chercher_client (arg :nom,prénom,adresse : ci);
          if not trouvé
          then
               créer_client (arg : nom,prénom,adresse :
               ci,dernier_numéro_client);
          fi;
          od
     else
          test_correspondance;
     fi;
     envoyer numéro_client : ci to "enreg_cmde";
od <idclient>;
```

test_correspondance;

```
if (nom,prénom : ci) = (nom,prénom : fichier client)
then
     if  adresse : ci not = adresse : fichier client
     then
          modifier_client(arg : adresse : ci)
     fi
else
     créer_client(arg : nom,prénom,adresse :
     ci,dernier_numéro_client);
fi;
```

```
<enreg_cmde>


do
recevoir numéro_client from <idclient>;
refus:=false;
créer commande_client;
for each LC = ligne_commande-client from commande_client
     do
     if numéro_produit : LC not = blank
     then
          do
          chercher_produit(arg : numéro_produit : LC);
          if trouvé
          then
               do
               if libellé : LC not = libellé : fichier produit
               then
                    do
                    chercher_produit(arg : libellé : LC);
                    if not trouvé
                    then
                         do
                         envoyer bon_commande_refusé
                         to <traitement_cas_litigieux>;
                         refus:=true;
                         od;
                    fi;
                    od;
               fi;
               créer ligne_commande_client;
               calcul montant_total;
               od
          else
               do
               chercher_produit(arg : libellé : LC);
               if trouvé
               then
                    do
                    créer ligne_commande_client;
                    calcul montant_total;
                    od
               else
                    do
                    envoyer bon-commande_refusé
                    to <traitement_cas_litigieux>;
                    od;
               fi;
               od;
          fi
          od
     else
          do
          chercher_produit(arg : libellé : LC);
```

```
            if trouvé
            then
                 do
                 créer ligne_commande_client;
                 calcul montant_total;
                 od
            else
                 do
                 envoyer bon_commande_refusé
                 to <traitement_cas_litigieux>;
                 od;
            fi;
            od;
       fi;
       od LC;

       if refus = true
       then
            envoyer bon_commande_refusé
            to <traitement_cas_litigieux>
       else
            if montant_total refusé
            then
                 envoyer bon_commande_refusé
                 to <traitement_cas_litigieux>
            else
                 envoyer numéro_commande to <majmoins>;
            fi;
       fi;
od <enreg_cmde>;
```

5.2.4. Validation.

Test black box.

cfr annexe 1.

Test white box.

Nous n'incluons pas dans ce mémoire les tests "white box" de nos programmes, car cela nous aurait pris trop de temps. De plus, un tel test ne peut se faire que lorsque l'algorithme est presque au point.

5.2.5. Evaluation des algorithmes.

A ce niveau, nous pouvons établir une évaluation des algorithmes du point de vue coût d'accès à la base de données. Connaissant les statistiques de vente et d'approvisionnement de la firme Petitpas, nous pouvons établir le nombre moyen d'accès qu'un algorithme effectuera par jour. Une telle évaluation permettrait de choisir à notre niveau d'analyse entre plusieurs algorithmes. En réalisant Petitpas, notre but n'était pas tant la recherche de la performance que permettre la validation des outils que nous avons réalisé; en conséquence, nous n'avons pas recherché l'algorithme le plus performant.

CHAPITRE 6 :

IMPLEMENTATION DE L'APPLICATION ET DETERMINATION DES PRIMITIVES.

Jean-Pierre Bodart

6. Implémentation de l'application et détermination des primitives.

6.1. Implémentation des fichiers.

6.1.1. Détermination des fichiers physiques.

L'analyse fonctionnelle et l'analyse organique nous ont permis de déterminer les types d'article suivants :

- "CLIENT"
- "FOURNISSEUR"
- "COMMANDE-CLIENT"
- "CMDE-CLIENT-DIFF"
- "LIGNE-COMMANDE-CLIENT"
- "PRODUIT"
- "EXPEDITION"
- "LIGNE-D-EXPEDITION"
- "FICHE-DE-REAPPROVISIONNEMENT"
- "LIGNE-DE-FICHE-DE-REAPP"

Les critères ayant amenés à cette élaboration des types d'article sont les suivants :

- regroupement des données en un ensemble d'entités ayant des liens logiques cohérents du point de vue de la sémantique,
- regroupement selon les traitements.

A partir des résultats des analyses fonctionnelle et organique, voici notre choix d'implémentation des fichiers physiques. Nous avons déterminé dix fichiers qui sont :

- "CLIENT"
- "FOURNISSEUR"
- "COMMANDE-CLIENT"
- "CMDE-CLIENT-DIFF"
- "LIGNE-COMMANDE-CLIENT"
- "PRODUIT"
- "EXPEDITION"
- "LIGNE-D-EXPEDITION"
- "FICHE-DE-REAPPROVISIONNEMENT"
- "LIGNE-DE-FICHE-DE-REAPP"

Ce choix respecte les critères suivants :

- éviter de regrouper des fichiers de mouvements et permanents,
- respect d'une granularité acceptable permettant de respecter les contraintes.

Nous avons introduit le fichier "CMDE-CLIENT-DIFF", parce qu'il existe un traitement spécial sur les commandes qui sont différées (choisir des commandes différées lors d'un réapprovisionnement). Ce traitement étant séquentiel, nous avons préféré séparer les commandes différées des autres commandes en vue d'une optimisation. La nature du traitement détermine la nature du fichier (implémenter le fichier avec accès par clé ou séquentiel).

Nous avons choisi d'associer un fichier par type d'article, car les primitives que nous avons réalisées ne permettent pas d'associer plusieurs types d'article à un même fichier. Nous nous sommes limités à la réalisation de primitives de gestion associant un type d'article par fichier. La complexité de ces primitives serait plus importante pour assurer la gestion des données et la sécurité résultant de l'association de plusieurs types d'article à un fichier serait plus faible; toutes ces raisons ont déterminé notre choix.

6.1.2. Liste par fichier des fonctions nécessaires.

"CLIENT" :

- consultation aléatoire,
- insertion,
- modification.

"FOURNISSEUR" :

- insertion,
- modification.

"COMMANDE-CLIENT" :

- insertion,
- modification.

"LIGNE-COMMANDE-CLIENT" :

- consultation aléatoire,
- consultation séquentielle,
- insertion,
- modification.

"PRODUIT" :

- consultation aléatoire,
- modification.

"CMDE-CLIENT-DIFF" :

- consultation aléatoire,
- consultation séquentielle,
- suppression.

"EXPEDITION" :

- insertion.

"LIGNE-D-EXPEDITION" :

- insertion.

"FICHE-DE-RE. ?PROVISIONNEMENT" :

- insertion,
- consultation aléatoire.

"LIGNE-DE-FICHE-DE-REAPP" :

- insertion,
- consultation aléatoire,
- consultation séquentielle.

6.1.3. Description des fichiers et liste des accès associés.

Remarque liminaire.

Le formalisme utilisé pour la description des articles est celui du langage Pascal.

Types élémentaires utilisés dans les descriptions d'articles.

```
{ Entier strictement positif }
sposint = 1 .. 32767;
{ Entier positif }
posint = 0 .. 32767;
{ Nom }
typenom = packed array[1 ..30] of char;
{ Prénom }
typeprenom = packed array[1 .. 16] of char;
{ Adresse }
```

```
typeadr = packed array[1 .. 60] of char;
{ Date }
typedate = packed array[1 ..6] of char;
{ Libellé }
typelib = typenom;
{ Type de mesure }
typemes = typelib;
{ Motif de refus }
typemotif = typelib;
```

a) Fichiers des clients : fclient.

Structure d'un article.

```
client  = record
          { Numéro du client }
          numero  : sposint;
          { Nom du client }
          nom : typenom;
          { Prénom du client }
          prenom : typeprenom;
          { Adresse du client }
          adresse : typeadr;
end;
```

Accès.

- par le numéro de client :
  - identifiant,
  - numérique entier positif,
- par le nom de client :
  - non identifiant,
  - alphanumérique,

b) Fichiers des fournisseurs : ffourns.

Structure d'un article.

```
fourn = record
          numero : posint;
          nom : typenom;
          adresse : typeadr;
end;
```

Accès.

- par le numéro de fournisseur :
  - identifiant,
  - numérique entier positif,

c) Fichiers des commandes clients : fcmdecli.

Structure d'un article.

```
cmdecli = record
        { Numéro de commande }
        numero  : sposint;
        { Date de la commande }
        date : typedate;
        { Etat de la commande }
        etat    : char;
        { Numéro du client }
        num_cli : sposint;
end;
```

Accès.

- par le numéro de client :

  - identifiant,
  - numérique entier positif,

d) Fichiers des lignes de commande client : lcmdecli.

Structure d'un article.

```
cmdecli = record
        { Numéro de commande }
        numero  : sposint;
        { Date de la commande }
        date : typedate;
        { Etat de la commande }
        etat    : char;
        { Numéro du client }
        num_cli : sposint;
end;
```

Accès.

- par le numéro de commande client :

  - non identifiant,
  - numérique entier positif,

e) Fichiers des commandes différées : fcmdedif.

Structure d'un article.

```
cmdedif = record
        { Numéro de la commande différée }
        numero  : sposint;
end;
```

Accès.

- séquentiel,

f) Fichiers des produits : fproduit.

Structure d'un article.

```
produit = record
        { Numéro de produit }
        numero  : sposint;
        { Libellé du produit }
        libelle : typelib;
        { Prix unitaire du produit}
        prix    : integer;
        { Poids unitaire du produit }
        poids   : sposint;
        { Quantité en commande }
        qte_cmde : posint;
        { Quantité livrée par défaut }
        qte_liv_def : sposint;
        { Quantité en stock }
        qte_stk : posint;
        { Indicateur d'épuisement }
        ind_epui : char;
        { Quantité différée }
        qte_dif : posint;
        { Quantité économique de commande }
        qte_eco_cmde : sposint;
        { Point de commande }
        pt_cmde : sposint;
        { Unite de mesure }
        unit_mes : typemes;
end;
```

Accès.

- par le numéro de produit :
  - identifiant,
  - numérique entier positif,
- par le nom de produit :
  - identifiant,
  - alphanumérique,

g) Fichiers des expéditions : fexped.

Structure d'un article.

Remarque.

Ce fichier peut être facilement reconstitué à partir des lignes d'expédition, c'est la raison pour laquelle ce fichier n'existe pas physiquement.

Accès.

- séquentiel,

h) Fichiers des lignes d'expédition : fligexp.

Structure d'un article.

```
ligexp = record
        { Numéro de la commande du client }
        num_cmde : sposint;
        { Numéro du produit }
        num_prod : sposint;
        { Quantité commandée }
        qte_cmde : posint;
        { Quantité due }
        qte_due : posint;
        { Quantité livrée }
        qte_livree : posint;
        { Numéro du client }
        num_cli : sposint;
end;
```

Accès.

- par le numéro d'expédition :

  - non identifiant,
  - numérique entier positif,

i) Fichier des fiches de réapprovisionnement : fficreap.

Structure d'un article.

```
FR_type = record
        { Numéro de fiche de réapprovisionnement }
        no_fiche : posint;
        { Numéro du fournisseur }
        no_fourn : posint;
        { Nom du fournisseur }
        nom_fourn : typenom;
        { Adresse du fournisseur }
        adr_fourn : typeadr;
        { Date de réception, c'est-à-dire de
          livraison }
        date_livr : typedate;
        { Référence au bon de livraison }
        no_BL : posint;
end;
```

Accès.

- par le numéro de fiche de réapprovisionnement :

- identifiant,
- numérique entier positif,

i) Fichier des lignes de fiche de réapprovisionnement : fligfic.

Structure d'un article.

```
LFR_type = record
        { Numéro de fiche de réapprovisionnement }
        no_reapp : posint;
        { Référence interne = numéro du produit }
        ref_int : posint;
        { Référence externe }
        ref_ext : posint;
        { Nom du produit }
        nom_prod : typelib;
        { Quantité reçue }
        qte_recue : posint;
        { Quantité refusée }
        qte_ref : posint;
        { Motif du refus }
        motif_ref : typmotif;
        { Numéro de la commande génératrice }
        no_cmde : posint;
end;
```

Accès.

- par le numéro de fiche de réapprovisionnement,
  - non identifiant,
  - numérique entier positif,

6.2. Implémentation de l'architecture.

6.2.1. Architecture du point de vue statique.

La mise en oeuvre de l'architecture se scinde en trois parties : la constructiion des programmes, la synchronisation et la communication de données.

a) Construction de programmes.

Notre principe de construction fut d'associer un programme à un module de l'analyse organique. Ainsi, nous gardions le parallélisme optimum obtenu lors de l'analyse organique.

b) Communication de données.

Posons que :

le module_1 remplit les fonctions :

- identification du client,
- constitution de la commande du client,
- enregistrement de la commande du client,

- le module_2 remplit la fonction de "maj-",

- le module_3 remplit la fonction d'enregistrement de la livraison d'un fournisseur,

- le module_4 remplit la fonction de "maj+",

- le module_5 remplit la fonction de sélection des commandes différées.

Les communications de messages sont les suivantes :

De : module_1
Vers : module_2
Message : un "numéro de commande client"

De : module_3
Vers : module_4
Message : un "numéro de fiche de réapprovisionnement"

De : module_4
Vers : module_5
Message : un numéro de produit"

De : module_5
Vers : module_2
Message : un "numéro de commande différée"

A chaque communication de données entre programmes, nous avons associé une boîte aux lettres. Nous rappelons qu'un des objectifs qui nous étaient imposés, était de construire une architecture qui permette que plusieurs utilisateurs d'un terminal puissent simultanément effectuer le même travail à savoir soit l'"enregistrement d'une commande client", soit l'"enregistrement de livraisons fournisseurs". Leur nombre étant quelconque, il n'était pas raisonnable d'associer une boîte aux lettres par communication et par version d'un tel programme avec les programmes à version unique. Nous avons décidé que tous les programmes à nombre variable de versions dans le système communiqueraient via la même boîte aux lettres .

Nous avons donc les boîtes aux lettres suivantes :

- entre le module_1 et le module_2 : CJ_box,
- entre le module_3 et le module_4 : FR_box,
- entre le module_4 et le module_5 : P_box,
- entre le module_5 et le module_2 : CD_box,

c) Synchronisation.

Après avoir décrit le problème de la transmission de messages, il est nécessaire de définir un protocole permettant la synchronisation entre écrivains et lecteurs.

Pour les écrivains, il n'existe pas de problèmes; ils écriront quand cela sera nécessaire, seul un sémaphore est à prévoir : ce qui est réalisé au niveau de PUTMESS et GETMESS. En effet, l'écriture et la lecture dans une boîte aux lettres sont par définition des opérations mutuellement exclusives; enconséquence, nous avons décidé d'adjoindre à la fonction de base de ces primitives des fonctions de blocage et de déblocage du sémaphore associé à la boîte aux lettres afin d'assurer le déroulement continu de l'opération de lecture ou d'écriture.

Par contre, un lecteur ne doit lire un message que si un écrivain a déjà déposé un message dans la boîte aux lettres. Sous Unix, la solution idéale serait d'utiliser l'appel système SIGNAL pour prévenir le lecteur de l'existence d'un message.

Cet appel peut être employé dans le cas Petitpas, il est en effet possible d'envoyer un signal d'un processus attaché à un terminal à un autre processus attaché à un terminal différent, à condition que les deux processus travaillent sous le même nom d'utilisateur et si on connaît le numéro du processus.

Puisque pour Petitpas, il existera un seul nom d'utilisateur, il serait donc possible d'utiliser cet appel système. Mais un autre problème se pose. en effet, il n'existe pas de file d'attente où l'on pourrait stocker les signaux. Dès lors, un écrivain ne peut envoyer un signal à un lecteur que si ce dernier n'est pas actif, sinon le signal stopperait le lecteur avant qu'il ne puisse traiter complètement le message en cours. Il serait donc nécessaire que l'écrivain attende la fin du travail du lecteur avant d'envoyer son signal; ce qui nous semble inacceptable.

En attendant une généralisation de cet appel système, la solution choisie sera que le processus lecteur devra boucler sur un ordre de lecture, mais pour éviter que ce processus ne travaille continuellement, il s'endormira pendant quelques instants si aucun message n'est disponoble.

Présentation de la solution.

Le lecteur exécutera les opérations suivantes :

```
étiquette : existe-t-il un message?
                    (ceci est réalisé grâce à la procédure
                     GETMESS [cfr annexe 2])
            sinon dormir (ceci est réalisé grâce à la procédure
                          DORMEZ)
                  aller en étiquette
            si oui le traiter
```

aller en étiquette.

Liste des sémaphores utilisés.

Les boîtes aux lettres étant partagées entre plusieurs programmes, nous avons assigné un sémaphore par boîte aux lettres; ce qui nous permet de garantir qu'il n'y aura pas de perte d'informations.

Voici la liste des sémaphores utilisés :

- le sémaphore s_CJ_box est associé à la boîte CJ_box;
- le sémaphore s_FR_box est associé à la boîte FR_box;
- le sémaphore s_P_box est associé à la boîte P_box;
- le sémaphore s_CD_box est associé à la boîte CD_box;

6.2.2. Architecture du point de vue dynamique.

Nous distinguons deux types de programme dans notre architecture :

- les programmes module_1et module_3 d'une part,
- les programmes module_2, module_4 et module_5 d'autre part.

Les premiers sont des programmes à copies multiples tandis que les seconds sont à version unique dans le système Petitpas. Nous parlerons également de programmes amonts et avals; ceci faisant référence à la relation qui existe entre les programmes d'un même flux de données. Dans l'architecture qui nous occupe, nous remarquons deux flux : le flux des commandes clients et celui des livraisons fournisseurs.

Le système des programmes de Petitpas est géré grâce à l'utilisation d'un fichier appelé "fstasys" donnant l'état du système; ce fichier contient les noms des programmes actifs dans le système Petitpas et un compteur de copies par nom de programme.

a) Démarrage.

Le fait de lancer l'exécution d'un programme à copies multiples entraîne la vérification du fichier d'état et sa mise à jour qui consistent d'une part à vérifier que le programme aval direct à copie simple lui correspondant est présent dans le système Petitpas ou s'il ne l'est pas à lancer son exécution et d'autre part à incrémenter le compteur associé au nom du programme amont lui-même d'une unité.

Le programme aval à son tour analyse le fichier d'état, il incrémente le compteur qui lui est associé d'une unité et vérifie que son programme aval direct existe sinon il lance l'exécution de ce dernier; ainsi le lancement des programmes se déroule suivant un déclenchement en cascade.

b) Arrêt.

Lors de la terminaison d'un programme amont, le compteur associé à ce type de programme amont est décrémenté d'une unité par le programme qui se termine. L'arrêt de tous les programmes (processus) est effectif dès lors que les programmes avals n'ont plus de données à traiter et que les compteurs des programmes amonts sont nuls. Dans les faits, les programmes avals n'ayant plus de données à traiter consultent le fichier "fstasys", et si le compteur associé à leur type de programme amont est nul, ils se terminent.

6.3. Langage de programmation.

Comme nous l'avons déjà précisé le langage de programmation nous a été imposé en vue de juger de son aptitude pour des applications de gestion : il s'agit du langage Pascal. Nous avons déjà signalé par ailleurs [1] qu'un langage de programmation adapté à la gestion doit permettre la structuration des données notamment celles rangées dans des fichiers, il doit permettre également la structuration des actions pour faciliter la maintenance ultérieure.

a) Structuration des données.

Ce langage offre les types de données classiques à savoir : "char", "integer", "real", "array", "boolean", "pointer", "file", mais en plus il possède le type composé "record", lequel est constitué d'éléments de type simple ou composé; donc la représentation que peut réaliser Pascal de la réalité est plus grande. A cette possibilité de structuration, s'ajoute la possibilité pour l'utilisateur de se définir ses propres types de données, d'où un accroissement de représentativité des données.

b) Manipulation de fichiers.

Pascal considère un fichier comme une suite de données. L'accès à ces données est uniquement de type séquentiel.

c) Structuration des actions.

Le Pascal possède des instructions de structuration telles que :

- "repeat ... until ..."
- "while ... do ..."
- "for ... do ..."
- "if ...
  then ..."

---

[1] Rapport sur le Pascal
Jean-Pierre Bodart et Jean-Paul Tixhon
Octobre 1980

else ... "
- "case of ... "
- la notion de bloc "begin ...end" ou plus précisément la notion d'instruction composée.

En plus, ce langage donne la possibilité de construire des procédures et des fonctions. Ces fonctions et procédures peuvent comporter des variables locales dont l'avantage est de pouvoir circonscrire facilement la portée d'une variable. Elles permettent également le passage de paramètres de deux manières :

- par valeur,
- par nom, il est donc possible en Pascal de passer comme paramètre un nom de procédure ou de fonction, ce qui n'est pas permis dans certains langages de programmation; il en découle une grande souplesse d'utilisation. Le Pascal Unix dispose évidemment de toutes les facilités décrites dans ce paragraphe.

d) Conclusion.

La structuration des données et la structuration des actions offertes par le langage Pascal constituent des avantages certains, tandis que l'accès séquentiel aux fichiers étant le seul accès prévu constitue le point faible.

Dans des applications par lots, cet inconvénient n'apparaîtra pas, tandis que dans des applications conversationnelles, il constituera un handicap important. C'est la raison pour laquelle nous avons dû réaliser un système complet d'accès aux fichiers pour pouvoir réaliser l'application Petitpas qui est du type conversationnel.

Quant au Pascal Unix dont nous disposions, il respecte le Pascal standard en ce qui concerne les trois points cités plus haut et donc hérite des mêmes avantages et inconvénient.

6.4. Primitives nécessaires.

6.4.1. Primitives d'accès aux fichiers.

Le paragraphe 6.1 nous a permis de mettre en évidence la nécessité de disposer de fonctions de consultation aléatoire, de consultation séquentielle, d'insertion, de modification et de suppression d'article.

Les primitives sont donc :

- consult
- readnext
- insert
- modify

- delete

6.4.2. Primitives de communication de donnees.

Les primitives necessaires doivent permettre l'envoi et la reception de messages; soient les primitives "putmess" et "getmess".

6.4.3. Primitives de synchronisation au niveau des donnees.

Les primitives de synchronisation necessaires pour gerer la concurrence en demande d'une ressource doivent permettre le blocage ou le deblocage de cette ressource. Ce blocage-deblocage est realise a l'aide de semaphores, en fait ce sont de simples verrous. Les primitives necessaires sont donc "lock" et "unlock".

6.4.4. Primitives de mise a jour du fichier d'etat du systeme Petitpas.

- givecpt : permet d'obtenir la valeur du compteur de copies associe a un programme de l'architecture,

- majcpt : permet l'incrementation ou la decrementation de la valeur du compteur de copies associe a un programme de l'architecture.

6.4.5. Primitives de manipulation de processus.

Definition.

Un processus est un programme en execution.

- dormez : provoque l'endormissement du processus qui s'execute,
- executez : permet de lancer l'execution d'un programme a partir d'un processus.

6.4.6. Primitives de manipulation de l'ecran des terminaux.

Afin que le dialogue entre l'utilisateur et la machine paraisse agreable a ce dernier, nous avons realise l'application Petitpas en faisant de la manipulation d'ecran. Les primitives de manipulation d'ecran n'etant disponibles que pour les terminaux Digital VT100, l'application Petitpas ne peut

fonctionner correctement que sur ces terminaux. Pour une documentation complète de ces primitives se rèfèrer au document U9.

CHAPITRE 7 : DESCRIPTION DES PRIMITIVES.

Jean-Paul Tixhon

7. Description des primitives.

7.1 Gestion de fichier

7.1.1 Analyse fonctionnelle

En tentant de réaliser une implémentation de l'application Petitpas sous le système UNIX et en PASCAL,nous sommes parvenus à déterminer les outils indispensables dont nous devions disposer.

Une des caractéristiques fondamentales d'une application de gestion du type Petitpas est de nécessiter une base de données(ce terme étant pris au sens large).

L'outil le plus important est donc, celui que nous allons décrire dans ce septième chapitre,il s'agit d'un ensemble de primitives gérant les accès aux fichiers.

Nous avons tenter de mettre en oeuvre un système minimal qui permet d'exécuter les différents accès rencontrés dans Petitpas. Notre but n'est certainement pas de réaliser un système de gestion de base de données complet,mais bien un outil simple et dont le temps de réalisation ne devait pas excéder quelques mois.

Deux de nos contraintes étaient le système d'exploitation UNIX et le langage PASCAL.
Or pour UNIX,les fichiers sont une suite de caractères,il n'existe pas de notion d'article.Trois primitives sont disponibles : la lecture de caractères (read),l'écriture de caractères (write),le positionnement à un endroit du fichier (seek).Pour plus de détail,on se réfèrera au chapitre deux du mémoire.

En PASCAL standard,le seul type de fichier existant est le fichier séquentiel sur lequel on ne peut que lire ou écrire séquentiellement des articles.

Quelles sont les caractéristiques de notre système ?

Il permet sous un système d'exploitation UNIX et à partir du langage PASCAL, l'accès direct par clé identifiante ou non

identifiante, par clé numérique ou alphanumérique,et l'accès séquentiel à un fichier.

Ce système ne comprend pas la synchronisation des accès entre plusieurs processus vers un même fichier.

Nous définissons (dans le formalisme de Bacchus) un type d'article et ses accès de la manière suivante :

< déclaration de type d'article > ::= < identificateur de type d'article > < ensemble d'ordres >

< ensemble d'ordres > ::= < ordre > | < ordre > < ensemble d'ordres >

< ordre > ::= séquentiel | < clé d'accès >

< clé d'accès > ::= < identificateur d'attribut de l'article >

Pour déterminer les opérations de manipulations d'article,nous nous sommes également basés sur [A9].

Il nous semble indispensable de disposer des primitives suivantes :

Remarques :

Dans la description des primitives,nous parlons de mécanisme de recherche, il s'agit d'un mécanisme permettant de déterminer un ensemble d'articles du fichier .Par exemple,un mécanisme pourrait être : déterminer un article à partir d'une valeur de clé d'accès identifiante.
Chaque primitive indique à l'utilisateur la réussite ou l'échec de l'opération, mais nous ne l'indiquons pas explicitement.

a) Primitive d'accès au premier article

nom : CONSULT

arguments:
f : type d'article
o : ordre
m : mécanisme de recherche

fonction :
si f désigne un type d'article,si o est un ordre défini sur f ,et s'il existe des articles cibles de f associés au mécanisme de recherche m, fournir le premier de ceux-ci.

b) Primitive d'accès à l'article suivant

nom : READNEXT

arguments :
f : type d'article
o : ordre
m : mécanisme d'accès
p : article précédent
fonction :
si f désigne un type d'article,si o est un ordre défini sur f,s'il existe des articles cibles de f associés au mécanisme de recherche m,si p est du nombre et n'est pas le dernier,fournir parmi ces articles celui qui suit p.

c) Primitive de suppression d'un article

nom : DELETE

arguments :
f : type d'article
ar : article
fonction :
si f est un type d'article,ar un article de ce type alors supprimer l'article ar du type d'article f.

d) Primitive d'insertion d'un article dans un fichier

nom : INSERT

arguments :
f : type d'article
ar : article
fonction :
si f est un type d'article,ar un article de ce type alors ajouter ar dans f en respectant les ordres déclarés.

e) Primitive de modification de valeurs d'item d'un article

nom : MODIFY

arguments :
f : type d'article
ar : article
{ (at,valat) } : ensemble de couples (attribut,valeur d'attribut)
fonction :
si f est un type d'article,ar un article de ce type,alors

pour tous les couples (at,valat) tel que at est un attribut de ar,remplacer la valeur existante de at par la valeur valat.

7.1.2 Analyse organique.

La liste des primitives que nous avons écrites en PASCAL découle directement de celle qui a été proposée au point 7.1.1 . Mais aux primitives DELETE,INSERT,MODIFY,CONSULT,READNEXT nous avons ajouté des primitives plus "technologiques" : OPENF,CLOSEF,CREATEF,OPENAC,CLOSEAC, CREATEAC,DELETEAC.

Voyons maintenant ce que vont réaliser ces primitives.

a) Création d'un fichier données et ouverture (CREATEF).

Cette procédure a pour but de créer un fichier de données et de l'ouvrir à partir du renseignement suivant :

- nom du fichier de données.

En retour,si l'opération a réussi nous disposerons d'un descripteur de fichier.

b) Ouverture d'un fichier données (OPENF).

Cette procédure ouvre un fichier de données à partir du renseignement suivant :

- nom du fichier de données.

En retour,si l'opération a réussi nous disposerons d'un descripteur de fichier.

c) Fermeture d'un fichier données (CLOSEF).

Cette procédure ferme un fichier de données ouvert à partir du renseignement suivant :

- descripteur de fichier de données.

d) Création d'un accès (CREATEAC).

Cette procédure permet de créer un accès par clé ou séquentiel, à partir des renseignements suivants :

- descripteur du fichier de données
- description de la clé :
  sa position dans l'article et sa longueur

pour un accès séquentiel,la position et la longueur égaleront zéro.

En retour,si l'opération a réussi,la procédure donne un descripteur d'accès.

e) Ouverture d'un accès (OPENAC).

Cette procédure permet d'ouvrir un accès ,à partir des renseignements suivants:

- descripteur du fichier de données
- description de la clé(sa position dans l'article et sa longueur).

En retour,si l'opération a réussi,la procédure donne un descripteur d'accès.

f) Fermeture d'un accès (CLOSEAC).

Cette procédure permet de fermer un accès défini,à partir des renseignements suivants:
- descripteur du fichier de données
- descripteur de l'accès.

g) Suppression d'un accès (DELETEAC).

Cette procédure permet de supprimer un accès défini,à partir des renseignements suivants:
- descripteur du fichier de données
- descripteur de l'accès.

h) Consultation du fichier données (CONSULT).

Cette procédure permet de consulter un fichier de données, à partir des renseignements suivants:
- descripteur du fichier de données
- descripteur de l'accès désiré
- une valeur de clé.

En retour,si l'opération a réussi,la procédure renvoie l'article (dans le cas d'une clé identifiante) ou le premier article (dans le cas d'une clé non identifiante) ayant cette clé.

i) Insertion d'un article dans le fichier données (INSERT).

Cette procédure permet d'insérer un article dans le fichier de données, à partir des renseignements suivants:
- descripteur du fichier données
- l'article à insérer.

De plus la procédure effectue la mise-à-jour de tout les accès définis .

j) Suppression d'un article du fichier données (DELETE).

Cette procédure permet de supprimer un ensemble d'articles du fichier de données, à partir des renseignements suivants:
- descripteur du fichier données
- descripteur de l'accès à utiliser
- un valeur de clé.

De plus,la procédure effectuera une mise-à-jour de tous les types d'accès définis .
Si le type d'accès est non identifiant,la procédure refusera

de supprimer un article.

k) Lecture sèquentielle d'un fichier de donnèes (READNEXT).

Cette procèdure lit l'article suivant du fichier donnèes selon l'accès dèsirè(sèquentiel normal ou sèquentiel sur une clè).
La procèdure nècessite les renseignements suivants:

- descripteur du fichier donnèes
- descripteur de l'accès.

En retour,si l'opèration a rèussi,la procèdure envoie l'article lu.

l) Modification d'un article (MODIFY).

Cette procèdure permet de modifier l'article courant du fichier de donnèes,à partir des renseignements suivants:

- descripteur du fichier de donnèes
- descripteur de l'accès
- l'article (complet) modifiè.

Cette procèdure traitant l'article courant,elle se situera aprés une lecture (READNEXT ou CONSULT).
Si une clè de l'article a èté modifièe la procèdure refusera d'exècuter l'opèration.

Remarque :

Pour les spècifications prècises,la conception,les jeux de tests,on se rèfèrera à l'annexe 2.

1) Description des fichiers et du mécanisme de recherche.

Notre système comporte deux types de fichier : le fichier données et le fichier index.

Le fichier données est une suite chronologique d'articles. Ces articles sont de longueur fixe.

Le fichier index est une suite,à tout moment triée par ordre croissant,des clés. Il existe un fichier index par accès par clé défini . Un article de ce fichier index contient une valeur de clé et un pointeur vers un article du fichier données.Pour les valeurs de clés non identifiantes, nous aurons plusieurs articles reprenant cette valeur avec des pointeurs différents. Une clé est définie par sa position dans l'article et sa longueur,si l'accès est séquentiel,la position et la longueur seront égales à zéro.

Exemple :

Si l'on considère un fichier client et que l'on désire accéder à ce fichier de manière séquentielle,par le nom du client(clé non identifiante), par le numéro du client(clé identifiante),nous aurons le schéma suivant :

- un fichier données client

- deux fichiers index pour l'accès par le nom et pour l'accès par le numéro.

Pour les clients suivants :

NOM	NUMERO
Dupont	2
Durand	1
Dupont	3

Nous aurons les fichiers suivants : fichier données

NOM	NUMERO
Dupont	2
Durand	1
Dupont	3

fichier index,accès sur le nom	clé	pointeur
	Dupont	1
	Dupont	3
	Durand	2

fichier index,accès sur le numéro	clé	pointeur
	1	2
	2	1
	3	3

Le mécanisme général de recherche est donc évident,à partir d'une valeur de clé nous recherchons sa première équivalence dans le fichier index correspondant à l'accès désiré.Si cette clé existe,on atteint l'article du fichier données grâce au pointeur.

Pour l'accès séquentiel,on lit séquentiellement le fichier de données.

Dans la présentation des primitives,nous parlons de descripteur de fichier données ou d'accès. Ces descripteurs contiennent des informations nécessaires aux primitives.

Pour le fichier données,le descripteur contient :

- la longueur de l'article
- le nombre d'articles du fichier
- la liste des accès définis sur le fichier
  - la position de la clé dans l'article
  - la longueur de la clé.

Pour le fichier index,le descripteur contient :

- la position de la clé dans l'article
- la longueur de la clé
- un indicateur d'accès identifiant ou non
- le type de la clé (alphanumérique ou numérique).

2) Description de l'architecture.

Les douze primitives qui sont à la disposition de l'utilisateur font appel à d'autres procédures de plus bas niveau et ce pour mieux maîtriser la complexité des problèmes,pour localiser au maximum les dépendances liées à l'implémentation et pour éviter d'écrire plusieurs fois certaines séquences d'instructions.

Nous avons ainsi les procédures suivantes :

a) Création d'un fichier index (CREATIN).

Cette procédure permet de créer un fichier index à partir du fichier données et cela pour une nouvelle clé définie.

b) Génération d'un nom de fichier index (GEN).

c) Lecture du descripteur de fichier données (GETFDON).

d) Ecriture du descripteur de fichier données (PUTFDON).

e) Positionnement au premier article du fichier données (POSFDON).

f) Lecture du descripteur d'accès (GETFIND).

g) Ecriture du descripteur d'accès (PUTFIND).

h) Positionnement à la première clé d'un accès (POSFIND).

i) Lecture d'un article du fichier données (GETDATA).

j) Ecriture d'un article du fichier données (PUTDATA).

k) Lecture de l'article précédant du fichier données (GETPDATA).

L) lecture d'un article clé du fichier index (GETKEY).

m) Ecriture d'un article clé du fichier index (PUTKEY).

n) Ajoût d'une clé dans un index (AJOUTER).

o) Recherche de la première occurrence d'une valeur de clé dans un index (SEARCHFIRST).

p) Comparaison de deux clès (COMPARE).

q) Comparaison de deux clès entières (TEST_INT).

7.1.3 Implémentation

Ce chapitre a pour but de donner d'une part certains choix d'implémentation et d'autre part les limites technologiques de notre système.

L'accès direct par clé a été réalisé par la technique de la recherche dichotomique.Cette technique permet,en effet,un accès séquentiel aisé et un accès direct dont l'algorithme de réalisation n'est pas complexe.

Cette recherche dichotomique s'effectue sur le fichier, car cela pouvait être réalisé dans le temps imparti, mais il est évident que pour améliorer les performances du système,il serait utile d'effectuer la recherche dans une table en mémoire centrale.En réalisant nos primitives,nous nous sommes efforcés de localiser dans certaines procédures ce choix pour limiter les changements.

Nous avons décidé de ne pas supprimer physiquement les articles. Ceux-ci se ont étiquettés et resteront dans le fichier,mais nous metton à la disposition de l'utilisateur un outil permettant le comp ctage de fichier (pour la description de ce compactage,on se reportera au manuel d'utilisation).

Comme nous l'avons décrit précédemment,notre système comporte deux types de fichiers.Comment sont-ils physiquement organisés ?

Le fichier données est constitué en premier lieu du descripteur de fichier,puis de la suite des articles données.Le dernier élément de cette suite est un article spécial de fin de fichier.

L'article données sera le suivant :

- un flag
  - s = supprimé
  - p = présent
  - f = fin de fichier
- l'article,en cas de fin de fichier cet article est à la valeur maximale.

Le fichier index est constitué de la même façon.L'article index sera le suivant :

- un flag (voir plus haut)
- le pointeur,numéro relatif de l'article données

- la clé(idem plus haut en cas de fin de fichier).

Voici dans un formalisme PASCAL,les descripteurs complets et les articles physiques :

```
constantes :
        { longueur d'un bloc }
        lbloc = 512;
        { longueur maximum d'un article logique }
        lmaxart = 510;
        { nombre maximum d'accès par fichier }
        maxacc = 10;
        { longueur maximum d'une clé }
        lmaxcle = 508;
        { indicatif de fin de fichier }
        feof = 'f';
        { indicatif d'article supprimé }
        fsup = 's';
        { indicatif d'article présent et vivant }
        fpres = 'p';
        { nombre maximum d'articles admis dans un fichier
          c-à-d. maxint - 2 }
        maxart = 32765;
types :

        { définition d'un type de zone }
        typezone = (C,E);

        { type buffer de la longueur d'un bloc }
        bufbloc = packed array [1..lbloc] of char;

        { article de données }
        data = packed array [1 .. lmaxart] of char;

        { article clé }
        key = packed array [1..lmaxcle] of char;

        { integer positif }
        posint = 0..maxint;

        { type servant pour les cles }
        nposint = -1..maxint;

        { descripteur de fichier UNIX  }
        fdok = -1..maxfich;


        { description d'une clé d'accès }
        cleacces = record
                { position de la clé dans l'article }
                poscle : nposint;
                { longueur de la clé }
                longcle : nposint;
                { type de clé }
                type_cle : typezone;
```

```
end;

descript = record
        { longueur de l'article }
        lart : posint;
        { nombre d'articles dans le fichier }
        nbart : posint;
        { nombre d'articles supprimes }
        artsup : posint;
        { longueur du descripteur en bytes}
        ldesdon:integer;
        { liste des clès d'accès définies }
        lsacces : array [1..maxacc] of cleacces;
end;

{ descripteur du fichier données }
filcont = record
        { nom du fichier }
        nom : string;
        { descripteur de fichier UNIX }
        fd : fdok;
        { header du descripteur de fichier }
        header:descript;
end;


{ descripteur d'accès }
filind = record
        { position de la clè dans l'article }
        poscle : nposint;
        { longueur de la clè en bytes }
        longcle : nposint;
        { accès identifiant }
        ac_ident:boolean;
        { type de clè }
        type_cle:typezone;
        { descripteur de fichier UNIX }
        fd : fdok;
        { longueur de l'article index }
        lart : posint;
        { nombre d'articles index }
        nbart : posint;
        { longueur du descripteur en bytes}
        ldesind:integer;
        { dernière clè de l'accès }
        lastkey : key;
end;
{ description de l'article index }
artind = record
        { flag de suppression }
        flagsup : char;
        { numèro de l'article dans
        le fichier de données }
        num_art:posint;
        { la clè }
```

```
        cle : key;
end;

{ description de l'article données }
artcont = record
          { flag de suppression }
          flagsup : char;
          { article }
          donnee : data;
          end;
```

Pour réaliser notre outil,nous nous sommes heurtés à quelques limites technologiques.

Nous nous sommes limités à des articles physiques de 512 caractères, donc notre article logique est au maximum de 510 caractères(512 - 2 caractères pour le flag).

La longueur maximum d'une clé est de 508(512 - 2 caractères pour le flag - 2 caractères pour le pointeur = 508 caractères).

Le nombre maximum d'accès possibles sur un fichier est de 10. En effet,par programme le nombre maximum de fichiers ouverts en même temps est ,sur UNIX,de 15,donc 15 - 3 fichiers standards - 1 fichier données - 1 descripteur utilisé par certaines primitives = 10.

La taille des fichiers données et index est limitée à MAXINT.En effet,lors du calcul de la position de début d'un article dans le fichier ([numéro relatif de l'article - 1] * longueur de l'article physique), nous aurions dû utiliser un résultat réel pour atteindre le chiffre de déplacement maximal possible.Avec ce résultat,nous devions ensuite calculer le déplacement en bloc et en byte,pour réaliser les opérations "seek" .Ces déplacements sont à indiquer en entier ,pour cela, il fallait utiliser la fonction PASCAL "trunc" qui transforme un réel en entier. Or il a été impossible de compiler les procédures utilisant "trunc".Le résultat du calcul de déplacement est donc entier,ce qui nous limite à une taille de fichier de MAXINT.

7.2 Communication de messages entre processus.

7.2.1 Analyse fonctionnelle.

Le second problème que nous avons abordé,concerne la communication de messages et la synchronisation de cette communication entre les modules de l'architecture de Petitpas.

L'outil que nous devions réaliser se rapproche d'un système "producteur-consommateur",où des processus producteurs envoient des messages à destination des processus consommateurs. Les processus producteurs doivent de plus prévenir les consommateurs de l'existence de messages.

Deux de nos contraintes étaient le langage PASCAL et le système d'exploitation UNIX.

PASCAL ne nous a pas imposé de contraintes pour réaliser cet outil,car ce langage ne prévoit rien sur la communication.

UNIX,par contre,offre à l'utilisateur un moyen de communication entre processus : le "pipe".Mais cet outil ne peut servir qu'entre un processus père et ses fils(voir chapitre 2). Vu notre volonté d'aborder le problème en toute généralité,il nous a donc été impossible de retenir cet appel système. Nous nous sommes tournés vers une autre méthode de résolution,que nous exposerons par la suite.

UNIX offre également un outil qui pourrait permettre à un processus producteur de prévenir un processus consommateur de l'existence d'un message à traiter,il s'agit des appels système "signal" et "kill"(voir chapitre 2). Mais,ici aussi,nous n'avons pas pu retenir ces appels système. En effet,il n'existe pas de files d'attente où l'on pourrait stocker les signaux. Dè lors,si un producteur prévient un consommateur,qui traite un message,ce producteur interrompt immédiatement ce consommateur.Le message traité est alors perdu. Il serait donc nécesaire que le producteur attente la fin du travail du consommateur avant d'envoyer son signal, solution que nous n'avons pas voulu retenir,car trop contraignante,pour notre système(voir chapitre 6.2).

Notre système ne comprend donc pas la synchronisation de cette communication. Mais nous avons tout de même trouver une manière de résoudre le problème,pour cela il est utile de lire le chapitre 6.2 . Dans ce chapitre,nous expliquons qu'un module consommateur de Petitpas peut si sa file de messages est vide,savoir si

les producteurs sont encore actifs. Si ces producteurs sont éteints alors le consommateur peut s'arrêter sinon il boucle sur la lecture de messages. Cette résolution n'a pas été intégrée dans l'outil de communication pour laisser libre choix à un programmeur.

Quelles sont les caractéristiques de notre outil ?

Il permet sur un système UNIX et à partir du langage PASCAL, la communication de messages entre un nombre quelconque de processus producteurs ou consommateurs.

Nous devons disposer des primitives suivantes :

a) Primitive d'écriture d'un message dans une boîte aux lettres

nom : PUTMESS

arguments :
b : boîte aux lettres
m : message à écrire
fonction :
si b est une boîte aux lettres,si m est un message,déposer m dans b.

b) Primitive de lecture d'un message dans une boîte aux lettres

nom : GETMESS

arguments :
b : boîte aux lettres
fonction :
si b est une boîte aux lettres,alors fournir m le premier message disponible.

7.2.2 Analyse organique.

La liste des primitives que nous avons écrites en PASCAL découle directement de celle qui a été proposée au point 7.2.1 . Mais aux primitives PUTMESS,GETMESS nous avons ajouté des primitives plus "technologiques" : CREATBOX,DELBOX.

Voyons maintenant ce que vont réaliser ces primitives.

Remarque :

Dans cette description,nous introduisons la notion de sémaphore. Deux processus ne peuvent travailler en même temps sur la même boîte aux lettres,il faut donc prévoir un moyen pour protéger celle-ci.Pour ce faire, nous avons utilisé les primitives de manipulation de sémaphore de UNIX (LOCK,LOCKIF,LOCKED, SEMOPEN,UNLOCK),primitives que nous avons transcrites en PASCAL.

a) Création d'une boîte aux lettres (CREATBOX).

Cette procédure a pour but de créer une boîte aux lettres, à partir du renseignement suivant :

- nom de la boîte.

La procédure garantit la création unique de la boîte.

b) Suppression d'une boîte aux lettres (DELBOX).

Cette procédure permet de supprimer une boîte aux lettres, à partir du renseignement suivant :

- nom de la boîte aux lettres.

La procédure supprimera la boîte même s'il existe encore des messages.

c) Lecture d'un message d'une boîte aux lettres (GETMESS).

Cette procédure permet de lire un message,à partir des renseignements suivants :

- le nom de la boîte aux lettres
- le sémaphore de la boîte.

En retour,la procédure envoie le premier message disponible,s'il en existe.

d) Ecriture d'un message dans une boîte aux lettres (PUTMESS).

Cette procédure permet d'écrire un message dans une boîte,s'il existe encore de la place pour ce message,à

partir des renseignements suivants :

- nom de la boîte aux lettres
- le sémaphore de la boîte
- le message à écrire.

Remarque :

Pour les spécifications, la conception, les jeux de tests, on se réfèrera à l'annexe 2.

1) Description de la boîte aux lettres et de la gestion des messages.

La boîte aux lettres est une liste circulaire de messages. Ces messages sont de longueur fixe. La gestion des messages est FIFO. Nous aurons comme pour les primitives d'accès aux fichiers, un descripteur de cette boîte.

Il contient les informations suivantes :

- le nombre de messages encore vivants
- un pointeur vers le premier message à lire
- un pointeur vers la première place libre
- le nombre maximum de messages que l'on peut déposer dans la boîte.

Le mécanisme de lecture est le suivant : s'il existe un message à lire (nombre de messages vivants > 0), le lire(pointeur en lecture) et mettre à jour les informations du descripteur.

Le mécanisme d'écriture est le suivant : s'il existe une place libre, écrire le nouveau message(pointeur en écriture) et mettre à jour les informations du descripteur.

2) Description de l'architecture.

Les quatre primitives qui sont à la disposition de l'utilisateur font appel à d'autres procédures de plus bas niveau et ce pour mieux maîtriser la complexité des problèmes, pour localiser au maximum les dépendances liées à l'implémentation et pour éviter d'écrire plusieurs fois certaines séquences d'instructions.

Nous avons ainsi les procédures suivantes :

a) Lecture du descripteur de boîte (GETFBOX).

b) Ecriture du descripteur de boite (PUTFBOX).

c) Lecture physique d'un message (RMESS).

d) Ecriture physique d'un message (WMESS).

7.2.3 Implèmentation.

La boîte aux lettres se trouve sur fichier.Elle contiendra en premier lieu le descripteur de boîte,puis la liste des messages.

Voici dans un formalisme PASCAL,le descripteur de boîte complet :

constantes :

```
{ longueur d'un bloc }
lbloc = 512;
{ longueur du descripteur de boite aux lettres}
ldesb = 10;
{ longueur maximum d'un fichier UNIX }
lfic = 1024;
```

types :

```
{ type buffer de la longueur d'un bloc }
bufbloc = packed array [1..lbloc] of char;

{ descripteur de fichier UNIX  }
fdok = -1..maxfich;

{ descripteur de boite aux lettres }
filbox = record
        { pointeur en ecriture }
        ptwrite : integer;
        { pointeur en lecture }
        ptread : integer;
        {nombre de message actif }
        nbre :integer;
        { longueur des messages }
        lmes : integer;
        { nombre maximum de messages de ce type
        pour la boite aux lettres }
        maxmes : integer;
end;
{ Dèclaration des types propres aux sèmaphores. }
verrou = (bloque, non_bloque);
path = string;
bsize = 1 .. 512;
fd = 0..14;
bsizeok = -1 .. 512;
```

Quelles sont les limites physiques de notre outil ?

Nous avons limitè la longueur des messages à 512 caractères.

La taille maximum de notre boîte est de 1024 caractères.

CHAPITRE 8 :

DESCRIPTION DE LA CONDUITE DU SYSTEME D'INFORMATION PETITPAS.

8. Description de la conduite du système d'informations Petitpas.

8.1. Démarrage et arrêt du système.

a) Pour le gestionnaire du système.

Le gestionnaire du système ne devra effectuer aucune procédure spéciale pour lancer l'application Petitpas,celle-ci peut démarrer dès que l'ordinateur est mis en route.

En fin de session,le gestionnaire du système devra :

a) vérifier que tous les programmes de l'application sont arrêtés,cela peut s'effectuer par la commande TESTSTOP,qui liste pour chaque programme de Petitpas le nombre de versions qui s'exécutent encore,

b) effectuer le compactage des fichiers (voir point 8.2.),

c) effectuer un sauvetage de la base de données(voir point 8.2.).

b) Pour l'utilisateur.

L'utilisateur peut accéder au système Petitpas en se connectant au système UNIX par la commande habituelle "login" suivie du nom "petitpas" et du mot de passe.

A ce moment,s'exécute automatiquement un programme de Petitpas (menu.p)et ce à la place du programme habituel demandant à l'utilisateur quel traitement il désire réaliser,et lancant l'exécution du programme correspondant au souhait exprimé(soit l'enregistrement d'une livraison fournisseur,soit l'enregistrement d'une commande client).

Pour l'arrêt,l'utilisateur indique au programme qu'il a terminé,celui-ci cloturant la session.

8.2. Maintenance du système.

La maintenance du système est de deux ordres : le nettoyage de la base de données,d'une part,et ,d'autre part,le sauvetage de cette base de données.

Nettoyage.

Comme nous l'avons dit dans le chapitre sept,notre système d'accès aux données ne supprime pas physiquement les articles.Dès lors,tous les soirs le gestionnaire de l'application Petitpas devra exécuter le programme MAINTFIC.

Ce programme trouve dans le fichier "flfile" le nom de l'ensemble des fichiers de l'application.Le programme demandera pour chaque fichier si le gestionnaire désire le compacter,de plus,si cette personne pourra proposer d'autres fichiers à inclure dans la liste.

A chaque fichier est associé un taux de compactage exprimé en pourcent, celui-ci indique le seuil d'articles supprimés à partir duquel il est nécessaire d'effectuer le compactage.

La structure de "flfile" est la suivante :

en premieu lieu le nombre d'entrées du fichier(un entier) puis la suite des articles du fichier.

```
tnomf = packed array[1..72] of char;
{ record de FLFILE }
entree = record
        { flag indiquant un fichier à compacter }
        flagsup : char;
        { taux de compactage en % }
        taux : integer;
        { nom du fichier }
        nom : tnomf;
end;
```

Sauvetage.

Chaque soir,le gestionnaire devra également effectuer un sauvetage de l'entièreté de la base de données.Cela pourra se faire par la commande BACKUP.Celle-ci copie tous les fichiers de l'application Petitpas et leurs index de /mnt/ps3/petitpas/database/data vers /mnt/ps3/petitpas/database/backup.

Une autre commande,RESTORE,permet de restaurer la base de données.

BACKUP et RESTORE effectuent le même travail avec les boîtes de communication du système Petitpas.

Ces commandes sont simplement deux commandes "shell" qui utilisent l'ordre "cp".

8.3. Architecture du système d'information Petitpas.

Le schèma du directoire "petitpas" est le suivant :

"bin" contient les versions objets des programmes,

"boxes" contient les boites des messages,

"database" contient les fichiers de l'application,

"src" contient les versions sources des programmes,

"update" contient les commandes de mise à jour du système.

Le schèma du directoire "bin" est le suivant :

(un nom débutant par "l" indique une librairie)

"laccess" contient les versions objets des programmes d'accès aux fichiers,

"application" contient les versions objets de l'application Petitpas,

"lcommunication" contient les versions objets des programmes de comunication de messages,

"lsynchro" contient les versions objets des programmes de synchronisation,

"lother" contient les versions objets de quelques autres primitives.

Le schèma du directoire "database" est le suivant :

"backup" contient le back-up des fichiers de l'application,

"data" contient les fichiers de l'application et les index.

Le schèma du directoire "src" est le suivant :

"access" contient les versions sources des programmes d'accès aux fichiers,

"application" contient les versions sources de l'application Petitpas,

"communication" contient les versions sources des programmes de comunication de messages,

"synchro" contient les versions sources des programmes de synchronisation,

"other" contient les versions sources de quelques autres primitives.

SUGGESTIONS ET CONCLUSIONS.

Suggestions et conclusions.

Par la réalisation d'une partie de l'application Petitpas sous UNIX, nous avons pu déterminer les outils dont devait disposer ce sysème d'exploitation pour être utilisé en informatique de gestion.

Nous avons dû concevoir et mettre en oeuvre les outils suivants : la gestion des accès aux fichiers, la communication de messages entre processus et la synchronisation entre processus.La réalisation de ces différents outils nous semblait indispensable,même si parfois UNIX présentait une solution à nos difficultés.

Tous les problèmes traités n'ont cependant pas toujours été résolus comme nous le désirions,certaines améliorations peuvent être apportées à notre travail.

Pour la gestion des accès aux fichiers,le problème concernant la protection des données est sujet à plusieurs améliorations :inclusion de cette protection dans l'outil proprement dit, réduction du grain de blocage au niveau de l'article et non pas du fichier.

De plus,le temps d'accès aux informations serait sensiblement diminuer en modifiant l'implémentation de la structure d'index.

Il en est de même pour la communication de messages entre processus, où la boîte aux lettres pourrait se situer en mémoire centrale.

Un des points que nous n'avons pu résoudre de manière suffisament générale,est celui qui concerne la synchronisation de processus,les appels système proposés par UNIX ne nous semblaient pas pouvoir répondre à notre préoccupation.

Le mécanisme de protection des fichiers que nous avons utilisé, les sémaphores,est également sujet à modifications.Les sémaphores, dont le nombre est restreint,sont considérés comme des ressources partagables entre plusieurs utilisateurs et non directement liées à la ressource qu'ils protègent,ce qui pourrait poser des problèmes si deux utilisateurs se servent d'un même

sémaphore.

Dans ce mémoire,nous avons adopté la méthode d'analyse enseignée à l'Institut d'Informatique,ce qui nous a donné la possibilité de l'appliquer dans le détail.Cette méthode a le mérite de couvrir tous les niveaux d'anlyse existants dans la résolution d'un problème et d'aboutir à une solution très structurée. Cette dernière particularité à l'avantage,entre autres, de faciliter la maintenance d'un projet.

Deux remarques nous sont cependant apparues lors de ce mémoire : il est parfois difficile de discerner les frontières qui existent entre les différentes étapes,d'une part, il est important qu'une méthode soit suffisament souple pour permettre les retours en arrière qui nous paraissent inévitables ,d'autre part.

BIBLIOGRAPHIE.

BIBLIOGRAPHIE

Analyse

[A1] Balzer N., Goldmann N.

Principle of good software specifications and their implications for specification language

Proceedings International Conference Specification of Reliable Software 1979

[A2] Bodart F.

Concepts, méthodes et outils de l'analyse de conception notes de cours de 3-ème licence

FUNDP 1980-1981

[A3] Bodart F.

Concepts, méthodes et outils de l'analyse fonctionnelle notes de cours de 1-ère licence

FUNDP 1978-1979

[A4] Bodart F. and Pigneur Y.

A model and a language for functional specification and evaluation of information system dynamics

Travaux de l'Institut d'informatique 1979

[A5] Bodart F. and Pigneur Y.

Dynamic Problem Specification Language : syntax

Travaux de l'Institut d'informatique 1979

[A6] Chen

An Entity/Relationship Model

ACM Transaction on Data Base System
Vol 1,n0 1 March 1976

[A7] Dijkstra

A discipline of programming

Prentice Hall,Englewood Cliffs 1976

[A8] Endres A.

An analysis of errors and their causes in
system programs

ACM Sigplan Notices (327-336) 1978

[A9] Hainaut JL.

Fichiers et banques de données
notes de cours de 1-ère et 2-ème licences

FUNDP 1978-1980

[A10] Hainaut JL.

Modèle Relationnel généralisé

FUNDP 1980

[A11] Hainaut JL.

Analyse organique du cas Petitpas
première partie : la base de données

FUNDP 1979-1980

[A12] Leheureux JM.,Hennebert AM.

Description de la firme Petitpas

Proposition d'automatisation

FUNDP 1978

[A13] Van Lamsweerde A.

Concepts,méthodes et outils de l'analyse organique
notes de cours de 2-ème licence

FUNDP 1979-1980

Pascal

[P1] Alabeau A., Figueras J., Pincon S.

PASCAL et les ordinateurs individuels

L'ordinateur individuel 13,60 1979

[P2] Cichelli R.J.

Fixing PASCAL's I/O

PASCAL newsletters 1980

[P3] Conradi R.

Further critical comments on PASCAL,
particulary as a system programming language

ACM-SIGPLAN 11.11,8 1976

[P4] Conway R., Gries D., Zimmerman E.C.

A primer on PASCAL
430 p

Winthrop Computer System Series 1976

[P5] David D.J., Deschamps J.L.

Programmer en PASCAL
160 p

Editions du P.S.I. Paris 1980

[P6] Disabeau C.

Presentation de PASCAL

L'ordinateur individuel 7,53 1979

[P7] Habermann A.N.

Critical comments on the programming
language PASCAL

ACTA INFORMATICA 3.1,45 1973

[P8] Jensen K., Wirth N.

Traduit de l'anglais par
Hernandez J.A., Kruchten Ph.

PASCAL:manuel de l'utilisateur
167 p

Eyrolles Paris 1980

[P9] Lang B.

Le langage PASCAL

Serie d'artic es dans MICRO-SYSTEMES
7,98(1979); 9,51 10,91 11,61(1980)

[P10] Lecarme D., Desjardins P.

More comments on the programming
language PASCAL

ACTA INFORMATICA 4,231 1975

[P11] Magnemat N., Thalmann D., Vaucher J.

Le langage PASCAL
330 p

Gaetan Morin Bordas-Dunod Bruxelles 1970

[P12] Monin Ph., Stamm M.

Libres questions a Niklaus Wirth
Le langage PASCAL

01 Informatique no130 mai 1979

Unix

[U1] Kernigham Brian W.

Unix For Beginners

Bell Laboratories
Murray Hill, New Jersey. 1975

[U2] Kernigham Brian W.

Unix Text Editor Tutorial

Bell Laboratories
Murray Hill, New Jersey. 1975

[U3] Kernigham Brian W.

Programming In C Tutorial

Bell Laboratories
Murray Hill, New Jersey. 1978

[U4] Lions J.

A comment on the Unix Operating System

Department of computer science
University of New South Wales 1975

[U5] Ritchie Dennis M.

The Unix I/O System

Bell Telephone Laboratories, N.J. 1975

[U6] Ritchie Dennis M.,Thompson Ken D.

The Unix Time Sharing System

Bell Laboratories
Murray Hill, New Jersey. 1978

[U7] Ritchie Dennis M.

On The Security Of Unix

Bell Laboratories
Murray Hill, New Jersey. 1975

[U8] Ritchie Dennis M.

C Reference Manual

Bell Laboratories
Murray Hill, New Jersey. 1975

[U9] Adans JP.,Vercheval J.

Mesures de performances de Unix
sous une charge universitaire(Mémoire)

FUNDP 1978-1979

[U10] Unix programmer's manual

FUNDP 1980-1981

GLOSSAIRE.

GLOSSAIRE.

annuaire : constitue un chemin d'accès vers les fichiers de tout type qu'il renferme.

catalogue : synonyme d'annuaire.

directoire : synonyme d'annuaire.

image : environnement d'exécution d'un programme.

processus : exécution d'une image de programme.

processus "ascendant" : processus "père" quelque soit son texte.

processus "descendant" : processus "fils" quelque soit son texte.

processus "enfant" : processus "fils" dont le texte peut avoir changé.

processus "fils" : processus créé par la primitive "fork".

processus "parent" : processus "père" dont le texte peut avoir changé.

processus "père" : est un processus qui crée un autre processus au moyen de la primitive "fork".

volume : sous-ensemble amovible du système de fichiers de Unix.

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Mémoire présenté par

Jean-Pierre Bodart
&
Jean-Paul Tixhon

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Mémoire présenté par

Jean-Pierre Bodart
&
Jean-Paul Tixhon

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

ETUDE DES POSSIBILITES DE UNIX APPLIQUEES A LA GESTION

(ANNEXES)

Mémoire présenté par

Jean-Pierre Bodart
&
Jean-Paul Tixhon

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

Année académique 1980-1981

TABLE DES MATIERES (ANNEXES)

Annexe 1

Annexe 2

Annexe 3

ANNEXES.

ANNEXE 1 : PETITPAS.

TRANSFORMATION D'UN MODELE ENTITE-ASSOCIATION EN MODELE BINAIRE.

TRANSFORMATION D'UN MODELE ENTITE-ASSOCIATION EN MODELE BINAIRE.

Principes de la transformation.

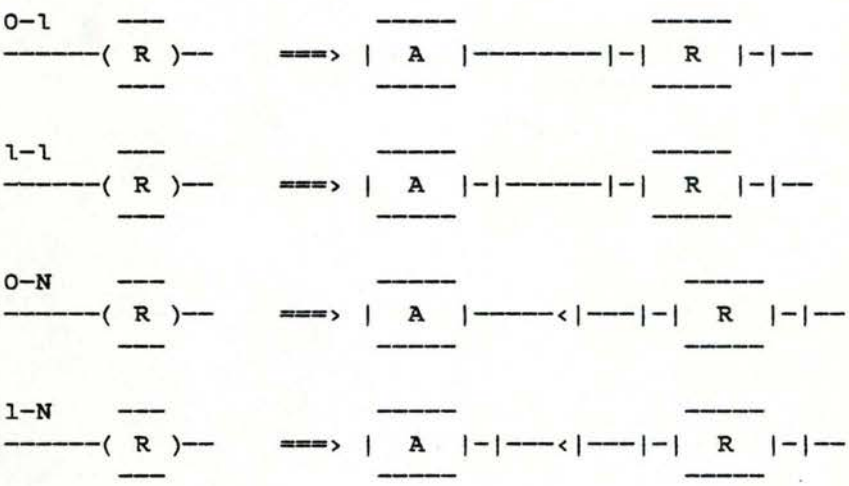
1. Les types d'entités sont conservés.

2. Les types d'association binaires sans attributs sont conservés sous forme de relations.

3. Un type d'association muni d'attributs est représenté par un type d'entité qui sera affecté de ces attributs; ce type d'entité est connecté aux anciens membres de l'association selon les règles suivantes :

   - chaque ancien membre est origine d'une relation 1-1 ou 1-N dont la cible est le nouveau type d'entité,

   - les propriétés d'existence et de connectivité sont transformées comme suit :



   - l'ensemble de ces nouvelles relations constitue l'identifiant du nouveau type d'entité.

4. Un type d'association de plus de deux membres est transformé comme au point 3.

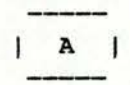
5. L'appartenance d'un attribut à un type d'entité est représenté par une relation entre cet attribut et ce type d'entité; les

propriétés d'existence et de cardinalité seront déduites du type d'attribut ainsi représenté.

6. D'autres contraintes d'intégrité peuvent encore être incluses (par exemple : identifiants de plusieurs relations).

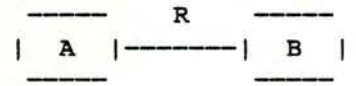
Conventions graphiques et représentation des contraintes.

1. Type d'entité de nom A :



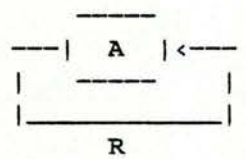
2. Attribut de nom B : B

3. Relation de nom R : entre A et B.



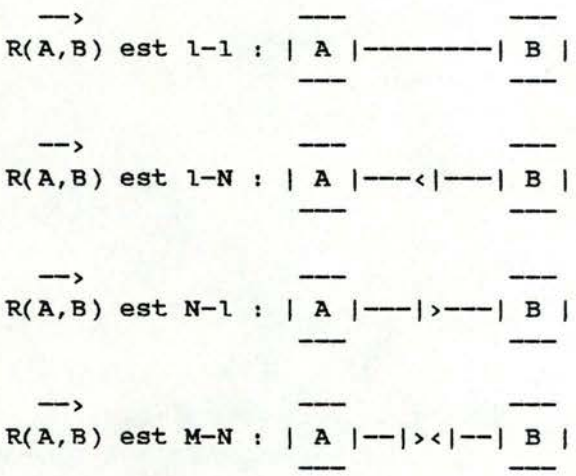
Remarque.
A ou B aurait pu être un attribut (sans case graphique).

4. Relation entre A et A :

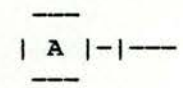


L'arc est orienté pour des raisons de désignation sémantique (dans PARENT(PERSONNE, PERSONNE) qui sont les enfants et les parents?); si R est symétrique (relation COLLEGUE, FRERE, CONJOINT) cette orientation est inutile.

5. Cardinalité (connectivité) d'une relation :



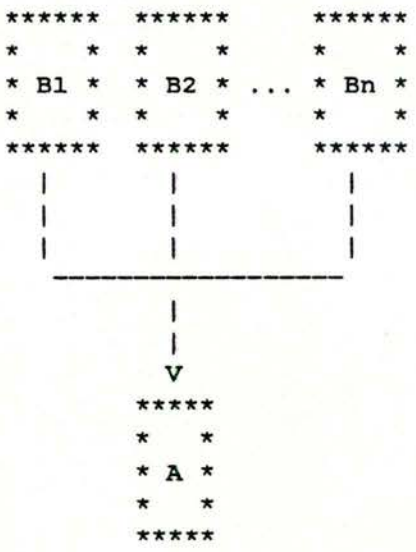
6. Propriété d'existence d'une relation : R est forte (obligatoire) pour A :



7. Identifiant d'un type d'entité A :

- tout B tel qu'il existe une relation R(A,B) N - 1 (ou 1 - 1 à fortiori),
- tout n-uple $B^1, B^2, \ldots, B^n$ tel qu'il existe des relations $R^1(A,B^1), R^2(A,B^2), \ldots, R^n(A,B^n)$ pour lesquelles :

  $|R^1[A,b^1] \; R^2[A,b^2] \cdot \ldots \cdot R^n[A,b^n]| <= 1$
  où $b^1 \in B^1, b^2 \in B^2, \ldots, b^n \in B^n$
  On notera : $id(A) = R^1, R^2, \ldots, R^n$.
  Ce cas se représentera graphiquement :



Remarque.

B1, B2, ..., Bn peuvent être attributs.

SPECIFICATION DES MODULES.

module ‹idclient›

Argument :

client : bon_commande_accepté.

Résultat :

numéro-client pour ‹enreg_cmde›.

Précondition :

néant.

Postcondition :

(client : bon_commande_accepté appartient fichier client) " \/" ((client : bon_commande_accepté n'appartient pas fichier client) "/\" (fichier client := fichier client + client : bon_commande_accepté)).

module ‹enreg_cmde›.

Argument :

numéro_client de ‹idclient›.

Résultat :

numéro_commande_client pour ‹majmoins›
ou bon_commande_refusé pour ‹traitement_cas_litigieux›.

Précondition :

numéro_client appartient fichier client.

Postcondition :

((commande acceptée) "/\" (fichier commande_client := fichier commande_client + bon_commande_accepté) "/\" (numéro_commande pour ‹majmoins›))
"\/" ((commande incorrecte) "/\" (bon_commande_refusé pour ‹traitement_cas_ litigieux›)).

Module d'enregistrement d'une livraison fournisseur.

Argument :

bon_de_livraison

Résultat :

fiche_de_réapprovisionnement

Pré-condition :

bon_de_livraison
"/\" (fournisseur "∈" {fournisseurs de Petitpas}
"\/" fournisseur "∈" {fournisseurs de Petitpas})
"/\" (numéro_de_fournisseur existe "\/" n'existe pas)
"/\" (numéro_de_commande existe "\/" n'existe pas)

Post-condition :

fiche_de_réapprovisionnement
"/\" numéro_de_fournisseur correct
"/\" numéro_de_commande correct
"\/" (fournisseur "∈" {fournisseurs de Petitpas} "/\" erreur)
"\/" (numéro_de_fournisseur incorrect "/\" erreur)
"\/" (numéro_de_commande incorrect "/\" erreur)

Module de 'maj+'.

Argument :

numéro_de_fiche_de_réapprovisionnement

Résultat : néant

Pré-condition : néant

Post-condition :

numéro_de_fiche_de_réapprovisionnement existe
"/\" "\/" produit_réapprovisionné METTRE-A-JOUR produit
"/\" sélection des commandes différées

Module de sélection des commandes différées.

Argument : numéro de produit

Résultat : numéro de commande différée

Pré-condition : néant

Post-condition : sélection "/\" ("$\nrightarrow$" (commande "$\rightarrow$" (une ligne de commande "$\rightarrow$" numéro de produit "/\" ligne de commande non satisfaite) "/\" numéro de commande différée)

Module de mise à jour moins.

Argument : numéro de commande

Résultat : néant

Pré-condition : néant

Post-condition : (ok "/\" (numéro de commande existe "/\" produit existe "/\" quantité de produit en stock non nulle)) "\/" quantité en stock nulle)

CONCEPTION DES MODULES.

```
<idclient>


do
     recevoir
      ci : client(nom,prénom,adresse,numéro) from terminal;
     chercher_client (arg : numéro : ci);
     if not trouvé
     then
          do
          chercher_client (arg :nom,prénom,adresse : ci);
          if not trouvé
          then
               créer_client   (arg   :    nom,prénom,adresse    :
               ci,dernier_numéro_client);
          fi;
          od
     else
          test_correspondance;
     fi;
     envoyer numéro_client : ci to "enreg_cmde";
od <idclient>;


test_correspondance;

if (nom,prénom : ci) = (nom,prénom : fichier client)
then
     if  adresse : ci not = adresse : fichier client
     then
          modifier_client(arg : adresse : ci)
     fi
else
     créer_client(arg        :        nom,prénom,adresse        :
     ci,dernier_numéro_client);
fi;
```

```
<enreg_cmde>


do
recevoir numèro_client from <idclient>;
refus:=false;
crêer commande_client;
for each LC = ligne_commande-client from commande_client
     do
     if numèro_produit : LC not = blank
     then
          do
          chercher_produit(arg : numèro_produit : LC);
          if trouvè
          then
               do
               if libellè : LC not = libellè : fichier produit
               then
                    do
                    chercher_produit(arg : libellè : LC);
                    if not trouvè
                    then
                         do
                         envoyer    bon_commande_refusè     to
                         <traitement_cas_litigieux>;
                         refus:=true;
                         od;
                    fi;
                    od;
               fi;
               crêer ligne_commande_client;
               calcul montant_total;
               od
          else
               do
               chercher_produit(arg : libellè : LC);
               if trouvè
               then
                    do
                    crêer ligne_commande_client;
                    calcul montant_total;
                    od
               else
                    do
                    envoyer       bon-commande_refusè       to
                    <traitement_cas_litigieux>;
                    od;
               fi;
               od;
          fi
          od
     else
          do
          chercher_produit(arg : libellè : LC);
```

```
            if trouvé
            then
                  do
                  créer ligne_commande_client;
                  calcul montant_total;
                  od
            else
                  do
                  envoyer          bon_commande_refusé          to
                  <traitement_cas_litigieux>;
                  od;
            fi;
            od;
      fi;
      od T.C;

      if refus = true
      then
            envoyer _______          bon_commande_refusé          to
            <traitement_cas_litigieux>
      else
            if montant_total refusé
            then
                  envoyer          bon_commande_refusé          to
                  <traitement_cas_litigieux>
            else
                  envoyer numéro_commande to <majmoins>;
            fi;
      fi;
od <enreg_cmde>;
```

```
‹enregistrement-livraison-fournisseur = ELF›


do
   recevoir un bon-de-livraison;
   for each BL = bon-de-livraison
   such that no_fourn (:BL) correct
   do
      for each LBL = ligne-bon-de-livraison
      do
         if no_produit (:LBL) correct
         then
              if qte_livrée (:LBL) > qte_cmdée
              then
                   do
                      envoyer une note-au-fournisseur;
                      refuser le surplus ;
                      qte_livrée := qte_cmdée;
                   od;
              fi;
         fi;
         créer une fiche-de-réapprovisionnement FR;
      od LBL;
      envoyer numéro-de-fiche-de-réapprovisionnement (:FR)
      à ‹maj+›;
   od BL;
od ELF;
```

```
<maj+>


do
   recevoir un numéro-de-fiche-de-réapprovisionnement
   de <enregistrement-livraison-fournisseur>;
   for FR = fiche-de-réapprovisionnement
   such that numéro (:FR) = numéro-de-fiche-de-réapprovisionnement
   do
      for each LFR = ligne-de-fiche-de-réapprovisionnement
      from FR
      do
         for P = produit
         such that no_produit (:P) = no_produit (:LFR)
         do
            qte_stk (:P) := qte_stk (:P) + qte (:LFR);
            qte_cmde (:P) := qte_cmde (:P) - qte (:LFR);
            envoyer num_pro (:P)
            à <sélection-des-commandes-différées>;
         od P;
      od LFR;
   od FR;
od maj+;
```

```
‹sélection-des-commandes-différées = SCD›


recevoir num_pro de ‹maj+›;
do
   for P = produit
   such that num_pro (:P) = num_pro
   do
      for each CD = commande-différée while qte_stk › 0
      do
         for each LCD = ligne-cmde-client-différée
         such that num_pro (:LCD) = num_pro (:P)
         do
            while qte_stk (:P) › 0
            do
               envoyer no_cmde (:CD) à ‹maj-›;
            od;
         od LCD;
      od CD;
   od P;
od SCD;
```

```
<maj_moins>


do
   while pas_reapp
   do
      traiter-une-commande-du-jour;
      if un-reapprovisionnment-effectue
      then
           while reapp-pas-fini
           do
              traiter-toutes-les-commandes-differees
           od;
      fi;
   od;
od;
```

```
<traiter-une-commande-du-jour>


recevoir no_produit;
for each C = commande-du-jour
do
   for each LC = ligne de commande du jour
   such that no_cmde (:LC) = no_cmde (:C)
   do
      for P = produit
      such that no_produit (:P) = no_produit
      do
         maj de P;
         créer une ligne d'expédition LE;
      od;
   od;
   créer une expédition E;
od;
```

‹traiter-toutes-les-commandes-différées›

```
do
   recevoir no_produit;
   for each CD = commande-différée
   do
      créer une expédition E;
      for each LCD = ligne-de-commande-différée
      from CD
      such that no_cmde (:LCD) = no_cmde (:CD)
      do
         if no_produit (:LCD) = no_produit
         and LCD non-apurée
         then
               do
                  maj LCD;
                  créer une ligne d'expédition LE (:E);
               od;
         fi;
      od;
   od;
od;
```

JEUX DE TESTS.

<idclient>

1) numèro_client :ci existe,client : ci = client :fichier client, adresse : fichier client correcte

2) numèro_client :ci existe,client : ci = client :fichier client, adresse : fichier client incorrecte

3) numèro_client : ci existe,client : ci non = client : fichier_client, nom-prènom : ci existe, adresse : fichier client correcte

4) numèro_client : ci existe,client : ci non = client : fichier_client, nom-prènom : ci existe, adresse : fichier client incorrecte

5) numèro_client : ci existe,client : ci non = client : fichier_client, nom-prènom : ci n'existe pas

6) numèro_client : ci n'existe pas,nom-prènom : ci existe, client : ci = client : fichier client, adresse : fichier client correcte

7) numèro_client : ci n'existe pas,nom-prènom : ci existe, client : ci = client : fichier client, adresse : fichier client incorrecte

8) numèro_client : ci n'existe pas,nom-prènom : ci existe, client : ci non = client : fichier client

9) numèro_client : ci n'existe pas,nom-prènom : ci n'existe pas.

<enregcmde>

1) numèro_produit : LC existe,produit : LC = produit : fichier produit, montants commande acceptès

2) numèro_produit : LC existe,produit : LC = produit : fichier produit, montants commande non acceptès

3) numèro_produit : LC existe,produit : LC non = produit : fichier produit, libellè : LC existe, montants commande acceptès

4) numèro_produit : LC existe,produit : LC non = produit : fichier produit, libellè : LC existe, montants commande non acceptès

5) numéro_produit : LC existe,produit : LC non = produit : fichier produit, libellé : LC n'existe pas

6) numéro_roduit : LC n'existe pas,libellé : LC existe, produit : LC = produit : fichier produit, montants commande acceptés

7) numéro_roduit : LC n'existe pas,libellé : LC existe, produit : LC = produit : fichier produit, montants commande non acceptés

8) numéro_roduit : LC n'existe pas,libellé : LC existe, produit : LC non = produit : fichier produit

9) numéro_produit : LC n'existe pas,libellé : LC n'existe pas.

<enrliv>

1) no_fourn (:BL) incorrect

2) no_fourn (:BL) correct; no_produit (:LBL) incorrect; nom_produit (:LBL) incorrect

3) no_fourn (:BL) correct; no_produit (:LBL) incorrect; nom_produit (:LBL) correct; qte_livree (:LBL) > qte_cmde

4) no_fourn (:BL) correct; no_produit (:LBL) incorrect; nom_produit (:LBL) correct; qte_livree (:LBL) <= qte_cmde

<majplus>

1) numéro (:FR) n'existe pas

2) numéro (:FR) existe; no_produit (:LFR) = no_produit (:P)

<scd>

1) N = numéro de produit reçu n'existe pas

2) N = numéro de produit reçu existe; num_pro (:LCD) = N; qte_stk (:P) > 0

3) N = numéro de produit reçu existe; num_pro (:LCD) = N; qte_stk (:P) = 0

<majmoins>

1) numéro de commande = num_cmde (:LC); num_pro (:LC) = numéro (:P); qte_cmde (:LC) > qte_stk (:P)

2) numéro de commande = num_cmde (:LC); num_pro (:LC) = numéro (:P); qte_cmde (:LC) > qte_stk (:P)

ANNEXE 2 : PRIMITIVES.

SPECIFICATIONS DES PRIMITIVES.

PRIMITIVES D'ACCES.

Nom : CREATEF

Arguments :
- en entrée :
  - un descripteur de fichier (FILCONT)
    + le nom du fichier (FILCONT.NOM)
    + la longueur de l'article donnée (FILCONT.HEADER.LART)
    (0 <= FILCONT.HEADER.LART <= LMAXART)
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier dont le champ NOM est un nom de fichier donné et LART la longueur d'un article de ce fichier, alors la fonction crée et ouvre le fichier de nom NOM, complète le descripteur de fichier FILCONT.
FILCONT.HEADER.NBART := FILCONT.HEADER.ARTSUP := 0
FILCONT.HEADER.LDESDON est calculé
Pour la liste des accès :
FILCONT.HEADER.LSACCES [I].POSCLE := -1
FILCONT.HEADER.LSACCES [I].LONGCLE := -1

Un résultat 0 indique un succès.

Si l'opération a échoué le résultat est :
1 si la longueur de l'article est incorrecte
2 si l'opération de création a échoué.

Nom : OPENF

Arguments :
- en entrée :
  - le descripteur de fichier (FILCONT) :
    + le nom du fichier (FILCONT.NOM)
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier dont le champ NOM est un nom de fichier donné, alors la fonction ouvre le fichier de nom NOM s'il existe ,et lit le descripteur FILCONT.

Un résultat de 0 indique un succès.

Si l'opération a échoué le résultat est :
1 en cas d'echec à l'ouverture.

Nom : CLOSEF

Arguments :
- en entrée :
  - descripteur de fichier complet (FILCONT)
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier donné complet, alors la fonction réécrit le descripteur(FILCONT) dans le fichier et ferme le fichier données .

Un résultat de 0 indique un succès.

Si l'opération a échoué le résultat est :
1 si la fermeture a échoué
2 si le fichier n'était pas ouvert.

Nom : CREATEAC

Arguments :

- en entrée :
  - descripteur de fichier complet(FILCONT)
  - descripteur d'accès (FILIND) :
    + position de la clé (FILIND.POSCLE)
    (0 <= FILIND.POSCLE <= FILCONT.HEADER.LART)
    + longueur de la clé (FILIND.LONGCLE)
    (0 <= FILIND.LONGCLE <= LMAXCLE)
    + indicateur d'accès identifiant (FILIND.AC_IDENT)
    + indicateur de type de clé (FILIND.TYPE_CLE)
    E : clé entière, C : clé alphabétique
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier complet et FILIND un descripteur d'accès complet,si le nombre d'accès définisable sur un fichier(MAXACC) n'est pas dépassé alors la fonction crée un accès pour le fichier données à partir des informations de FILIND(POSCLE et LONGCLE) et complète le descripteur d'accès.

Un résultat de 0 indique un succès.
Dans ce cas :

FILIND.LART est calculé (FILIND.LONGCLE + la longueur constante du reste des informations)
FILIND.NBART est calculé
FILIND.LDESIND est calculé

Si l'opération a échoué le résultat est :

1 si le fichier données n'est pas ouvert
2 si la clé est trop longue
3 si la position de la clé est incorrecte
4 si la clé se trouve en dehors de l'article (il faut FILIND.POSCLE + FILIND.LONGCLE <= FILCONT.HEADER.LART)
5 si le nombre d'accès défini est trop important(voir MAXACC)
6 si l'opération de création n'a pas réussi.
7 si l'accès existe déjà.

Nom : OPENAC

Arguments :
- en entrée :
  - descripteur de fichier complet (FILCONT)
  - descripteur d'accès (FILIND) :
    + position de la clé (FILIND.POSCLE)
    + longueur de la clé (FILIND.LONGCLE)
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier complet et FILIND un descripteur d'accès, alors si l'accès demandé existe pour le fichier données la fonction ouvre l'accès et lit le descripteur d'accès FILIND.

Un résultat de 0 indique un succès.

Si l'opération a échoué le résultat est :
1 si l'ouverture a échoué
2 si l'accès n'est pas défini
3 si le fichier données n'était pas ouvert.

Nom : CLOSEAC

Arguments :
- en entrée :
  - descripteur de fichier complet (FILCONT)
  - descripteur d'accès complet (FILIND)
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier complet et FILIND un descripteur d'accès complet, alors la fonction réécrit le descripteur d'accès(FILIND) et ferme le fichier index.

Un résultat de 0 indique un succès.

Si l'opération a échoué le résultat est :
1 si la fermeture a échoué
2 si le fichier données ou accès était non ouvert.

Nom : COMPACT

Arguments :

- en entrée :
  - le nom du fichier à compacter.
  - le taux d'articles supprimés
  - le terminal (fichier output)
- en sortie :
  - le résultat de l'opération

Fonction :

COMPACT compacte un fichier si le taux (rapport entre le nombre d'articles supprimés et le nombre total d'articles) est atteint.

Un résultat de 0 indique un succès.

Si l'opération a échoué, le résultat est :

1 si le compactage ne doit pas se faire (taux non atteint)

2 si une erreur s'est produite en manipulant le fichier

3 si une erreur s'est produite en manipulant les acces.

Nom : DELETEAC

Arguments :
- en entrée :
  - descripteur de fichier (FILCONT)
  - descripteur d'accès :
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de données complet et FILIND un descripteur d'accè défini pour ce fichier données,alors la fonction supprime l'accès.

Un résultat 0 indique un succès.

Si l'ope Si l'opération a échoué le résultat est 1.

Nom : CONSULT

Arguments :
- en entrée :
  - descripteur de fichier (FILCONT)
  - descripteur d'accès (FILIND)
  - une valeur de clé
- en sortie :
  - l'article lu
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier données, FILIND un descripteur d'accès et v une valeur de clé, alors la fonction renvoie le premier article correspondant à la valeur de v de l'accès FILIND du fichier données et positionne l'indicateur de succès à vrai en cas de succès; en cas d'échec,ne positionne que l'indicateur de succès à faux.

Nom : INSERT

Arguments :
- en entrée :
  - descripteur de fichier(FILCONT)
  - article à insérer
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier données et A un article, alors la fonction insère l'article dans le fichier données et effectue la mise à jour des accès définis en respectant l'ordre des clés.

Si l'opération a réussi le résultat est positionné à vrai et à faux sinon.

Nom : DELETE

Arguments :
- en entrée :
  - descripteur de fichier(FILCONT)
  - descripteur d'accès identifiant(FILIND)
  - valeur de clé
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier données, FILIND un descripteur d'accès identifiant et v une valeur de clé, alors la fonction supprime l'article du fichier données corespondant à cette valeur de clé et effectue la mise à jour des accès définis.

Si l'opération a réussi le résultat est positionné à vrai et à faux sinon.

Nom : READNEXT

Arguments :
- en entrée :
  - descripteur de fichier données (FILCONT)
  - descripteur d'accès (FILIND)
- en sortie :
  - article lu
  - indicateur de succès

Fonction :

Si FILCONT est un descripteur de fichier données et FILIND un descripteur d'accès, alors la fonction lit l'article suivant selon l'accès désiré.

Si l'opération a réussi le résultat est positionné à vrai et à faux sinon.

Nom : MODIFY

Arguments :
- en entrée :
  - descripteur de fichier(FILCONT)
  - descripteur d'accès(FILIND)
  - l'article complet modifié
- en sortie :
  - indicateur de succès

Fonction :

Si FILCONT est un descritpteur de fichier données, FILIND un descripteur d'accès et A l'article modifié, alors la fonction modifie l'article courant(lu par CONSULT ou READNEXT) avec la nouvelle valeur.

Si l'opération a réussi le résultat est positionné à vrai et à faux sinon.

Nom : CREATIN

Arguments :
- en entrée :
  - un descripteur de fichier données(FILCONT)
  - un descripteur d'accès(FILIND)
- en sortie :
  - indicateur de succès

Fonction :
CREATIN crée un fichier index selon l'accès demandé(FILIND.POSCLE, FILIND.LONGCLE) pour le fichier données.

Si l'opération a réussi le résultat est 0.
Dans ce cas :
FIND.NBART = nombre d'article index

Si l'opération a échoué le résultat est 1.

Nom : GEN

Arguments :
- en entrée :
  - le nom du fichier données
  - la place de l'accès dans la liste des accès (en fait,un chiffre de 0 à 99)
- en sortie :
  - le nom du fichier index

Fonction :
GEN génère le nom du fichier index à partir du nom du fichier données et de la place de l'index dans la liste des accès.
exemple : client,accès numéro 3 => client.03.

Nom : POSFDON

Arguments :
- en entrée :
  - le descripteur du fichier données(FILCONT)

Fonction :
POSFDON se positionne au premier article existant dans le fichier données.

Nom : POSFIND

Arguments :
- en entrée
  - le descripteur d'accès(FILIND)

Fonction :
POSFIND se positionne au premier article existant dans le fichier index.

Nom : PUTFDON

Arguments :
- en entrée :
  - le descripteur de fichier données(FILCONT)

Fonction :
PUTFDON écrit le descripteur du fichier données.

Nom : PUTFIND

Arguments :
- en entrée :
  - le descripteur d'accès(FILIND)

Fonction :
PUTFIND écrit le descripteur du fichier données.

Nom : GETFDON

Arguments :
- en sortie :
  - le descripteur de fichier données(FILCONT)

Fonction :
GETFDON lit le descripteur du fichier données.

Nom : GETFIND

Arguments :
- en sortie :
  - le descripteur d'accès(FILIND)

Fonction :
GETFIND lit le descripteur du fichier données.

Nom : GETKEY

Arguments :
- en entrée :
  - le descripteur d'accès(FILIND)
  - le numéro relatif de l'article à lire
- en sortie :
  - l'article index lu
  - le résultat de l'opération

Fonction :

GETKEY lit un article index grace à son numéro relatif dans le fichier.

Si l'opération a échoué le résultat est à faux.

Nom : GETDATA

Arguments :
- en entrée :
  - le descripteur du fichier données(FILCONT)
  - le numéro relatif de l'article à lire
- en sortie :
  - l'article données lu
  - le résultat de l'opération

Fonction :

GETDATA lit un article du fichier données à partir du numéro relatif dans le fichier.

Si l'opération a échoué le résultat de l'opération est à faux.

Nom : GETPDATA

Arguments :
- en entrée :
  - le descripteur du fichier données(FILCONT)
- en sortie :
  - l'article lu
  - le résultat de l'opération

Fonction :

GETPDATA lit l'article données qui précè de l'article courant du fichier données.

Si l'opération a échoué le résultat est à faux.

Nom : PUTKEY

Arguments :
- en entrée :
  - le descripteur d'accès(FILIND)
  - le numéro relatif de l'article
  - l'article à écrire
- en sortie :
  - le résultat de l'opération

Fonction :
PUTKEY écrit l'article index à la position relative spécifiée.

Si l'opération a échoué le résultat est à faux.

Nom : PUTDATA

Arguments :
- en entrée :
  - le descripteur du fichier données(FILCONT)
  - le numéro relatif de l'article
  - l'article à écrire
- en sortie :
  - le résultat de l'opération

Fonction :
PUTDATA écrit l'article données à la position relative spécifiée.

Si l'opération a échoué le résultat est à faux.

Nom : PUTPDATA

Arguments :
- en entrée :
  - le descripteur du fichier données(FILCONT)
  - l'article à écrire
- en sortie :
  - le résultat de l'opération

Fonction :
PUTPDATA écrit l'article données devant l'article courant.

Si l'opération a échoué le résultat est à faux.

Nom : COMPARE

Arguments :
- en entrée :
  - deux clés à comparer
  - l'adresse du premier byte à comparer pour les clés
  - la longueur des clés
  - le type des clés
- en sortie :
  - le résultat de la comparaison des clés

Fonction :

COMPARE compare deux clés et cela à partir des adresses des premiers bytes à comparer et suivant une longueur.

Le résultat est du type "typetest" : clé1 "lower" ou "equal" ou "greater" clé2.

Nom : TESTINT

Arguments :
- en entrée :
  - deux clés entières positives
- en sortie :
  - le résultat de la comparaison

Fonction :

TEST_INT compare deux clés entières positives.

Le résultat est du type "typetest" : clé1 "lower" ou "equal" ou "greater" clé2.

Nom : AJOUTER

Arguments :
- en entrée :
  - le descripteur d'accès (FILIND)
  - l'article index à ajouter
- en sortie :
  - le résultat de l'opération

Fonction :

AJOUTER ajoute une clé dans un fichier index en respectant l'ordre croissant des clés.

Si l'opération a échoué le résultat est à faux.

Nom : SEARCHFIRST

Arguments :
- en entrée :
  - le descripteur d'accès (FILIND)
  - la clé à rechercher
  - les bornes dans lesquelles il faut faire la recherche
- en sortie :
  - la position relative dans le fichier de la première occurrence de cette clé
  - la position relative de l'article données correspondant à cette clé
  - le résultat de l'opération

Fonction :

SEARCHFIRST recherche la première occurrence de la clé spécifiée dans le fichier index.

Si l'opération a échoué le résultat est à faux.

PRIMITIVES DE COMMUNICATION.

Nom : PUTMESS

Arguments :
- en entrée :
  - nom de la boîte aux lettres
  - le sémaphore associé à la boîte aux lettres
  - message à écrire
- en sortie :
  - résultat de l'opération

Fonction :

PUTMESS dépose un message dans la boîte aux lettres spécifiée dans la place pointée par FILBOX.PTWRITE, et effectue la mise à jour des informations du descripteur de boîte(FILBOX).
La gestion des messages est FIFO.

Le sémaphore est utilisé par la procédure pour protéger l'accès à la boîte pendant l'opération.

Si l'opération a réusssi le résultat renvoyé est 0.
Dans ce cas :
FILBOX.PTWRITE pointe vers la place suivante.
FILBOX.NBRE := FILBOX.NBRE + 1.

Si l'opération a échoué le résultat est :

1 si la boîte aux lettres n'existe pas
2 s'il est apparu un problème physique
3 s'il n'existe plus de place pour le message dans la boîte aux lettres.

Nom : GETMESS

Arguments :

-en entrée :

- nom de la boîte aux lettres
- le sémaphore associé à la boîte aux lettres

- en sortie :

- message lu
- résultat de l'opération

Fonction :

GETMESS lit le message pointé par FILBOX.PTREAD dans la boîte spécifiée et le détruit, la procédure effectue la mise à jour du descripteur de boîte(FILBOX).
La gestion des messages est FIFO.

Le sémaphore est utilisé par la procédure pour protéger l'accès à la boîte pendant l'opération.

Si l'opération a réussi le résultat renvoyé est 0.
Dans ce cas :

FILBOX.PTREAD pointe vers la place suivante.
FILBOX.NBRE := FILBOX.NBRE - 1.

Si l'opération a échoué le résultat est :

1 si la boîte aux lettres n'existe pas
2 s'il est apparu un problme physique
3 s'il n'existe plus de message dans la boîte aux lettres.

Nom : CREATBOX

Arguments :
- en entrée :
  - nom de la boîte aux lettres
  - longueur des messages que va contenir cette boîte (0 < longueur <= LBLOC)
- en sortie :
  - résultat de l'opération

Fonction :

CREATBOX crée une boîte aux lettres pour des messages de longueur fixe ,on doit indiquer la longueur de ces messages.

Si la boîte existe déjà,la procédure ne fait que l'ouvrir.

La procédure initialise le descripteur de boîte :
- FILBOX.PTWRITE:=FILBOX.PTREAD:= 1
- FILBOX.NBRE:= 0
- FILBOX.LMES est initialisé avec la longueur des messages
- FILBOX.MAXMES est calculé:
  LFIC / FILBOX.LMES

Si l'opération a réussi le résultat renvoyé est 0.

Si l'opération a échoué le résultat est :

1 si la boîte aux lettres existe déjà
2 si la longueur des messages est incorrecte
3 si la création a échoué.

Nom : DELBOX

Arguments :
-en entrée :
  -nom de la boîte aux lettres
-en sortie :
  - résultat de l'opération

Fonction :

DELBOX supprime une boîte aux lettres .

Si l'opération a réussi le résultat est 0.

Si l'opération a échoué le résultat est :

1 si la boîte aux lettres n'existe pas.

Nom : GETFBOX

Arguments :
- en entrée :
  - "file descriptor" UNIX du fichier
- en sortie :
  - le descripteur de boite aux lettres

Fonction :
GETFBOX lit dans le fichier boite aux lettres, le descripteur de la boite.

Nom : PUTFBOX

Arguments :
- en entrée :
  - "file descriptor" UNIX du fichier
  - le descripteur de boite aux lettres

Fonction :
PUTFBOX écrit dans le fichier boite aux lettres, le descripteur de boite.

Nom : RMESS

Arguments :
- en entrée :
  - "file descriptor" UNIX du fichier
  - le descripteur de boîte aux lettres
- en sortie :
  - le message lu
  - le résultat de l'opération

Fonction :

RMESS lit le message pointé par le pointeur en lecture (FILBOX.PTREAD) dans la boîte aux lettres.

Si l'opération a réussi le résultat est 0.

Si l'opération a échoué le résultat est -1.

Nom : WMESS

Arguments :
- en entrée :
  - "file descriptor" UNIX du fichier
  - le descripteur de boîte aux lettres
  - le message à écrire
- en sortie :
  - le résultat de l'opération

Fonction :

WMESS écrit le message pointé par le pointeur en écriture (FILBOX.PTWRITE) dans la boîte aux lettres.

Si l'opération a réussi le résultat est 0.

Si l'opération a échoué le résultat est -1.

PRIMITIVES SUR LES SEMAPHORES.

Nom : LOCK

Arguments :
- -en entrée :
  - le "file descriptor" du sémaphore
- -en sortie :
  - l'état du sémaphore avant intervention

Fonction :
Cette fonction verrouille un sémaphore, si celui-ci est non verrouillé, et attend pour le verrouiller jusqu'au moment du déverrouillage par un autre processus; de plus, elle renvoit l'état du sémaphore avant son intervention.

Nom : LOCKED

Arguments :
- -en entrée :
  - le "file descriptor" du sémaphore
- -en sortie :
  - l'état du sémaphore

Fonction :
Cette fonction permet simplement de connaître l'état d'un sémaphore.

Nom : LOCKIF

Arguments :
-en entrée :
- le "file descriptor" du sémaphore
-en sortie :
- l'état du sémaphore avant intervention

Fonction :
Cette fonction verrouille un sémaphore sans attente et renvoit l'état de ce dernier avant intervention.

Nom : SEMOPEN

Arguments :
-en entrée :
- le nom du sémaphore
-en sortie :
- le "file descriptor" du sémaphore

Fonction :
Fonction d'ouverture d'un sémaphore .

Nom : UNLOCK

Arguments :
-en entrée :
- le "file descriptor" du sémaphore
-en sortie :
- l'état du sémaphore avant intervention

Fonction :
Cette fonction déverrouille un sémaphore et renvoit l'état de celui-ci avant intervention.

PRIMITIVES DIVERSES.

Nom : CLRLINES

Arguments :
- en entrée :
  - le numéro de ligne début (> 0)
  - le numéro de ligne de fin (<= 24)

Fonction :

CLRLINES efface l'ensemble spécifié des lignes de l'écran.

Cette procédure ne marche que pour un terminal vt100 (procédure vt100).

Nom : READINT

Arguments :
- en entrée :
  - le terminal (fichier input)
- en sortie :
  - le nombre lu
  - le résultat de l'opération

Fonction :

READINT lit un nombre entier au terminal.

Si l'opération a échoué le résultat est à faux.

Nom : READPINT

Arguments :
- en entrée :
  - le terminal (fichier input)
- en sortie :
  - le nombre lu
  - le résultat de l'opération

Fonction :
READPINT lit un nombre entier positif au terminal.

Si l'opération a échoué le résultat est à faux.

Nom : READSTRING

Arguments :
- en entrée :
  - le terminal (fichier input)
  - la longueur de la chaîne de caractères a lire
- en sortie :
  - la chaîne de caractères

Fonction :
READSTRING lit au terminal une chaîne de caractères de longueur spécifiée ou inférieure.

Nom : COMPOST

Arguments :
- en entrée :
  - un nombre entier
  - un pas (nombre entier)
- en sortie :
  - le nombre entier + le pas

Fonction :
COMPOST sert essentiellement à effectuer le compostage d'une clé.

Nom : GIVECPT

Arguments :
- en entrée :
  - le nom du programme
- en sortie :
  - le nombre de versions de ce programmme qui s'exécutent

Fonction :
GIVECPT donne le nombre de versions d'un programme de Petitpas qui s'exécutent.

Si le programme n'existe pas le nombre est égal à zéro..

Nom : MAJCPT

Arguments :
- en entrée :
  - le nom du programme
  - un entier donnant le changement (+1 ou -1)

Fonction :
MAJCPT effectue la mise à jour (+ 1 ou -1) du nombre de versions d'un programme de Petitpas qui s'exécutent.

Nom : DORMEZ

Arguments :
- en entrée :
  - le temps de sommeil en secondes

Fonction :
DORMEZ met le processus en état de repos.

Nom : EXECUTEZ

Arguments :
- en entrée:
  - le nom du processus à exécuter

Fonction :
EXECUTEZ lance un processus.

CONCEPTION DE MODULES.

PRIMITIVES D'ACCES.

```
<closeac>


do
     recevoir un descripteur-de-fichier;
     recevoir un descripteur-d-accès;
     if fichier ouvert and
      accès ouvert then
          do
          écrire descripteur-d-acces;
          fermer acces;
          if fermeture réussie
          then
               envoyer indicateur := réussite
          else
               envoyer indicateur := échec;
          fi
          od
     else
          envoyer indicateur := échec;
     fi
od <closeac>;


<closef>


do
     recevoir descripteur-de-fichier;
     if fichier ouvert
     then
          do
          écrire descripteur-de-fichier;
          fermer fichier;
          if fermeture ok
          then
               envoyer indicateur := réussite
          else
               envoyer indicateur := échec;
          fi
          od
     else
          envoyer indicateur := échec;
     fi
od <closef>;
```

```
<compact>


do
     recevoir nom-de-fichier;
     recevoir taux-de-compactage;
     if taux reçus >= taux-fichier
     then
          do
          for tous les accès définis
          do
               compacter un accès;
          compacter fichier;
          envoyer indicateur := réussite;
          od
     else
          envoyer indicateur := pas de compactage;
     fi
od <compact>;
```

```
<createac>


do
     recevoir descripteur-de-fichier;
     recevoir descripteur-d-accès;
     if fichier ouvert and clé correcte and (nombre
     d'accès définis < maxacc)
     then
          do
          créer accès;
          écrire accès;
          envoyer indicateur := réussite;
          od
     else
          envoyer indicateur := échec;
     fi
od <createac>;


<createf>


do
     recevoir descripteur-de-fichier;
     if longueur record correcte
     then
          do
          créer fichier;
          if crétion correcte
          then
               do
               écrire descripteur-de-fichier;
               envoyer indicateur := réussite;
               od
          else
               envoyer indicateur := échec;
          fi
          od
     else
          envoyer indicateur := échec;
     fi
od <createf>;
```

```
<openac>


do
     recevoir descripteur-de-fichier;
     recevoir decripteur-d-accès;
     if fichier ouvert
     then
          do
          if clè d´accès dans liste des accès dèfinis
          then
               do
               ouvrir accès;
               lire descripteur-d-accès;
               envoyer indicateur := rèusssite;
               od
          else
               envoyer indicateur := èchec;
          fi
          od
     else
          envoyer indicateur := èchec;
     fi
od <openac>;


<openf>


do
     recevoir desripteur-de-fichier;
     ouvrir fichier;
     if ouverture correcte
     then
          do
          lire descripteur-de-fichier;
          envoyer indicateur := rèussite;
          od
     else
          envoyer indicateur := èchec;
     fi
od <openf>;
```

```
<consult>


do
   recevoir descripteur-de-fichier;
   recevoir descripteur-d'accès;
   recevoir valeur-de-clé;
   if descripteur-d'accès
   correspond à un accès dans lsacces du descripteur-de-fichier
   then
        do
           recherche dichotomique sur la valeur de clé;
           if recherche-fructueuse
           then
                while la clé précédente est la même
                do
                   lire la clé précédente;
                   renvoyer l'article et l'indicateur;
                od
           else
                renvoyer l'indicateur positionné à faux;
           fi;
        od
   else
        do
           renvoyer l'indicateur positionné à faux;
        od;
   fi;
od <consult>;
```

```
‹deleteac›


do
   recevoir un descripteur-de-fichier;
   recevoir un descripteur-d'accès;
   if descripteur-d'accès
   correspond à une clé dans lsacces
   then
        do
           supprimer l'accès;
           renvoyer l'indicateur positionné à faux;
        od
   else
        renvoyer l'indicateur positionné à faux;
   fi;
od ‹deleteac›;
```

```
‹delete›


do
   recevoir descripteur-de-fichier;
   recevoir descripteur-d'accès;
   recevoir une valeur-de-clé;
   if descripteur-d'accès
   correspond à un accès dans lsacces du descripteur-de-fichier
   then
        do
           recherche dichotomique sur la valeur-de-cle``;
           if recherche-fructueuse
           then
                do
                   supprimer les articles  correspondant  à  cette
                   clé
                   mettre-à-jour les autres index;
                od
           else
                renvoyer l'indicateur positionné à faux;
           fi;
        od
   else
        do
           renvoyer l'indicateur positionné à faux;
        od
   fi;
od ‹delete›;
```

‹insert›

```
do
   recevoir descripteur-de-fichier;
   recevoir descripteur-d'accès;
   if place-libre-dans-le-fichier
   then
        do
           ajouter l'article dans le fichier;
           incrémenter le nombre d'articles;
           mettre-à-jour les fichiers "index";
           renvoyer l'indicateur positionné à vrai;
        od
   else
        renvoyer l'indicateur positionné à faux;
   fi;
od ‹insert›;
```

<modify>

```
do
   recevoir un descripteur-de-fichier;
   recevoir un descripteur-d'accès;
   recevoir l'article modifié (complet);
   if descripteur-d'accès
   correspond à un accès dans lsacces du descripteur-de-fichier
   then
        do
           if aucune-clé modifiée
           then
                do
                   modifier l'article;
                   renvoyer l'indicateur de succès à vrai;
                od
           else
                renvoyer l'indicateur de succès à faux;
           fi;
        od
   else
        renvoyer l'indicateur de succès à faux;
   fi;
od <modify>;
```

```
‹readnext›


do
   recevoir descripteur-de-fichier;
   recevoir descripteur-d'accès;
   selon le descripteur-d'accès lire l'article suivant;
   if lecture-fructueuse
   then
        do
           renvoi de l'article lu;
           renvoi de l'indicateur de succès positionné à vrai;
        od
   else
        do
           renvoi de l'indicateur de succès positionné à faux;
        od
   fi;
od ‹readnext›;
```

PRIMITIVES DE COMMUNICATION.

```
<creatbox>


do
     recevoir nom-boîte;
     recevoir longueur-message;
     if longueur-message correcte
     then
          do
          créer boîte;
          envoyer indicateur := réussite;
          od;
     else
          envoyer indicateur := échec;
     fi;
od <creatbox>;


<delbox>


od
     recevoir nom-boîte;
     if boîte existe
     then
          do
          supprimer boîte;
          envoyer inicateur := réussite;
          od;
     else
          envoyer indicateur := échec;
     fi;
od <delbox>;
```

```
<getmess>


do
     recevoir nom-boîte;
     if boîte existe and existe un message
     then
          do
          lire le message;
          envoyer le message;
          envoyer indicateur := réussite;
          od;
     else
          envoyer indicateur := échec;
     fi;
od <getmess>;


<putmess>


do
     recevoir nom-boîte;
     recevoir message;
     if boîte existe and existe place
     then
          do
          écrire le message;
          envoyer le message;
          envoyer indicateur := réussite;
          od;
     else
          envoyer indicateur := échec;
     fi;
od <putmess>;
```

JEUX DE TESTS.

PRIMITIVES D'ACCES.

Remarque :
Pour chaque procédure, nous donnons une liste de tests, mais il est à remarquer que les jeux de tests doivent comprendre les diverses combinaisons possibles de ces jeux.

CLOSEAC

1) fichier non ouvert
2) accès non ouvert.

CLOSEF

1) fichier non ouvert.

COMPACT

1) fichier non existant
2) taux demandé < taux réel (pas de compactage)
3) incohérence fichier (nombre d'articles présents non = nombre d'articles trouvés)
4) incohérence accès (idem fichier et clé non présente dans un accès).

CONSULT

1) fichier non ouvert
2) accès non ouvert
3) la clé n'existe pas dans le fichier
4) l'article trouvé est supprimé.

CREATEAC

1) fichier non ouvert
2) clé de longueur incorrecte
3) position de la clé incorrecte
4) clé en dehors de l'article
5) maximum d'accès atteint.

CREATEF

1) longueur de l'article incorrecte.

DELETE

1) fichier non ouvert
2) accès non ouvert
3) accès non identifiant
4) clé non existante.

DELETEAC

1) fichier non ouvert
2) accès non existant.

INSERT

1) fichier non ouvert
2) fichier plein.

MODIFY

1) fichier non ouvert
2) accès non ouvert
3) modification d'une clé.

OPENAC

1) fichier non ouvert
2) accès non existant.

OPENF

1) fichier non existant.

PRIMITIVES DE COMMUNICATION.

CREATBOX

1) la boite existe déjà
2) la longueur des messages est incorrecte.

DELBOX

1) la boite n'existe pas.

GETMESS

1) la boite n'existe pas
2) la boite est vide.

PUTMESS

1) la boite n'existe pas
2) la boite est pleine.

MAPPING DES PRIMITIVES.

α Description des primitives.

Une description d'implémentation classique, sous la forme des cinq rubriques suivantes :

- arguments d'entrée,
- arguments de sortie,
- fonction réalisée par la primitive,
- liste des modules appelants,
- liste des modules appelés.

Les trois premières rubriques ayant déjà été mentionnées dans les descriptions fonctionnelle et organique des primitives, nous jugeons inutile de les mentionner à nouveau par soucis d'éviter les redondances; nous renvoyons donc le lecteur aux chapitres d'analyses fonctionnelle et organique pour cette partie de la description.

Quant aux deux dernières rubriques, elles n'ont pas encore été mentionnées et font l'objet de cette annexe. Leur utilité est de permettre d'abord la réalisation de la "cartographie" de l'architecture des primitives et ensuite de mieux percevoir les imbrications et dépendances.

Nous respectons ici la classification des primitives établie précédemment, à savoir que nous présentons d'abord les primitives d'accès et de manipulation des fichiers, ensuite celles permettant la communication de données entre processus, enfin celles permettant la synchronisation des processus.

PRIMITIVES D'ACCES.

- Primitives technologiques.

  - openf

    - modules appelants : insert,compact,
    - modules appelés : getfdon, posfdon, open,

  - closef

    - modules appelants : insert,compact,
    - modules appelés : close, putfdon,

- createf

  - modules appelants : compact,
  - modules appelés : creat, open, close, putdata, posfdon,

- openac

  - modules appelants : insert, delete,compact,
  - modules appelés : getfind, open, gen,

- closeac

  - modules appelants : insert, delete,compact,
  - modules appelés : putfind, close,

- createac

  - modules appelants : compact,
  - modules appelés : creat, close, open, putfind, creatin, gen,

- deleteac

  - modules appelants : compact,
  - modules appelés : unlink,gen,

- Primitives de manipulation.

  - insert

    - modules appelants : modules de programme d'application,
    - modules appelés : putdata, openac, ajouter, closeac, openf, closef.

  - consult

    - modules appelants : modules de programme d'application,
    - modules appelés : getdata, searchfirst.

  - readnext

    - modules appelants : modules de programme d'application,
    - modules appelés : getkey, getdata,

  - modify

    - modules appelants : modules de programme d'application,

- modules appelés : getpdata, putpdata, compare,

- delete

  - modules appelants : modules de programme d'application,
  - modules appelés : openac, closeac, getkey, putkey, getdata, putdata, compare, searchfirst,

- Primitive de maintenance.

  - compact

    - modules appelants : modules de programme d'application,
    - modules appelés : unlink, link, close, open, createf, openf, closef, gen, getdata, putdata, createac, deleteac, openac, closeac, getkey, putkey, compare, searchfirst,

- Primitives de niveau élémentaire.

  - ajouter

    - modules appelants : insert, creatin,
    - modules appelés : getkey, putkey, compare,

  - compare

    - modules appelants : ajouter, modify, delete, searchfirst,compact,
    - modules appelés : test_int,

  - searchfirst

    - modules appelants : consult, delete,compact,
    - modules appelés : getkey, compare,

  - test_int

    - modules appelants : compare,
    - modules appelés : néant,

  - getkey

    - modules appelants : readnext, delete, compact, searchfirst, creatin,
    - modules appelés : seek, uread,

  - putkey

- modules appelants : delete, compact, searchfirst, ajouter, creatin,
- modules appelés : seek, uwrite,

- getdata

  - modules appelants : consult, readnext, delete,
  - modules appelés : seek, uread,

- putdata

  - modules appelants : insert, delete, createf, compact,
  - modules appelés : seek, uwrite,

- getpdata

  - modules appelants : modify,
  - modules appelés : seek, uread,

- putpdata

  - modules appelants : modify,
  - modules appelés : seek, uwrite,

- creatin

  - modules appelants : createac,
  - modules appelés : getdata, putkey, posfind, posfdon, ajouter,

- gen

  - modules appelants : openac, openf, createac, createf, deleteac,
  - modules appelés : strlen,strbuf,strtobuf,

- getfdon

  - modules appelants : openf,createf,
  - modules appelés : seek, uread,

- getfind

  - modules appelants : openac,createac,
  - modules appelés : seek, uread,

- putfind

  - modules appelants : closeac,
  - modules appelés : seek, uwrite,

- putfdon

- modules appelants : closef,
- modules appelés : seek, uwrite,

- posfdon

  - modules appelants : openf,closef,createf,
  - modules appelés : seek,

- posfind

  - modules appelants : openac, closeac, createac, creatin,
  - modules appelés : seek,

Open, creat, close, seek, uread, uwrite, ... sont des primitives de Unix.

PRIMITIVES DE COMMUNICATION.

- Primitives technologiques.

  - creatbox

    - modules appelants : modules de programme d'application,
    - modules appelés : open, close, creat, putfbox,

  - delbox

    - modules appelants : modules de programme d'application,
    - modules appelés : unlink,

- Primitives de manipulation.

  - putmess

    - modules appelants : modules de programme d'application,
    - modules appelés : open, close, getfbox, putfbox, wmess,

  - getmess

    - modules appelants : modules de programme d'application,
    - modules appelés : open, close, getfbox, putfbox, rmess,

- Primitives de niveau élémentaire.

  - putfbox

    - modules appelants : creatbox, putmess, getmess,
    - modules appelés : seek, uwrite,

  - getfbox

    - modules appelants : putmess, getmess,
    - modules appelés : seek, uread,

- wmess

    - modules appelants : putmess,
    - modules appelés : seek, uwrite,

- rmess

    - modules appelants : getmess,
    - modules appelés : seek, uread,

Open, creat, close, unlink, seek, uread et uwrite sont des primitives de Unix.

PRIMITIVES DE SYNCHRONISATION.

- semopen
  - modules appelants : nèant,
  - modules appelès : open,
- lock
  - modules appelants : nèant,
  - modules appelès : uread,
- locked
  - modules appelants : nèant,
  - modules appelès : uread,
- lockif
  - modules appelants : nèant,
  - modules appelès : uread,
- unlock
  - modules appelants : nèant,
  - modules appelès : uread,

Open et uread sont des primitives de Unix.

PROGRAMMES.

PRIMITIVES D'ACCES.

Constantes.

```
{ longueur d'un bloc }
lbloc = 512;
{ longueur maximum d'un article logique }
lmaxart = 510;
{ nombre maximum de fichiers }
maxfich = 14;
{ longueur maximum d'un nom de fichier }
lnomfic = 14;
{ nombre maximum d'accès par fichier }
maxacc = 10;
{ longueur maximum d'une clé }
lmaxcle = 508;
{ indicatif de fin de fichier }
feof = 'f';
{ indicatif d'article supprimé }
fsup = 's';
{ indicatif d'article présent et vivant }
fpres = 'p';
{ longueur d'un string }
lstring = 72;
{ nombre maximum d'articles admis dans un fichier
   c-à-d. maxint - 2 }
maxart = 32765;
{ longueur de ce qui est constant dans filcont }
ldconst = 8;
{ longueur de ce qui est constant dans
   la liste des accès du descripteur }
ldlconst = 6;
{ longueur de ce qui est constant dans
   un descripteur d'accès }
ldiconst = 16;
{ longueur de ce qui est constant dans
   un article index }
laiconst = 4;
```

Types.

```
{ définition d'un type de zone }
typezone = (C,E);

{ valeurs possibles d'un octet }
```

```
octet = 0 .. 255;

{ valeurs d'une comparaison de caracteres }
typetest = (lower,equal,greater);

{ type buffer de la longueur d'un bloc }
bufbloc = packed array [1..lbloc] of char;

{ article de donnees }
data = packed array [1 .. lmaxart] of char;

{ article cle }
key = packed array [1..lmaxcle] of char;

ok = -1..0;

{ mode de creation d'un fichier }
modebits = (xhim,whim,rhim,xyou,wyou,ryou,xme,wme,
        rme,txt,sgid,suid,large,type1,type2,alloc);

{ type de SEEK }
seekreq = (abs,rel,plus,absb,relb,plusb);

{ mode d'ouverture }
openmode = (r,w,rw);

{ integer positif }
posint = 0..maxint;


{ type servant pour les cles }
nposint = -1..maxint;

{ descripteur de fichier UNIX  }
fdok = -1..maxfich;


{ description d'une cle d'acces }
cleacces = record
        { position de la cle dans l'article }
        poscle : nposint;
        { longueur de la cle }
        longcle : nposint;
        { type de cle }
        type_cle : typezone;
end;

descript = record
        { longueur de l'article }
        lart : posint;
        { nombre d'articles dans le fichier }
        nbart : posint;
        { nombre d'articles supprimes }
        artsup : posint;
        { longueur du descripteur en bytes}
```

```
        ldesdon:integer;
        { liste des clès d'accès définies }
        lsacces : array [1..maxacc] of cleacces;
end;

{ descripteur du fichier données }
filcont = record
        { nom du fichier }
        nom : string;
        { descripteur de fichier UNIX }
        fd : fdok;
        { header du descripteur de fichier }
        header:descript;
end;


{ descripteur d'accès }
filind = record
        { position de la clé dans l'article }
        poscle : nposint;
        { longueur de la clé en bytes }
        longcle : nposint;
        { accès identifiant }
        ac_ident:boolean;
        { type de clé }
        type_cle:typezone;
        { descripteur de fichier UNIX }
        fd : fdok;
        { longueur de l'article index }
        lart : posint;
        { nombre d'articles index }
        nbart : posint;
        { longueur du descripteur en bytes}
        ldesind:integer;
        { dernière clé de l'accès }
        lastkey : key;
end;
{ description de l'article index }
artind = record
        { flag de suppression }
        flagsup : char;
        { numèro de l'article dans
        le fichier de données }
        num_art:posint;
        { la clé }
        cle : key;
end;

{ description de l'article données }
artcont = record
          { flag de suppression }
          flagsup : char;
          { article }
          donnee : data;
          end;
```

Ajouter.p

```
#

{
*****************************************************************
*                                                               *
*                          "AJOUTER"                            *
* PROCEDURE D'AJOUT D'UNE VALEUR DE CLE DANS UN INDEX UTILISEE  *
* PAR LA PRIMITIVE : "INSERT"                                   *
*                                                               *
*****************************************************************
}

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Procédure de lecture d'un article "index" }
procedure getkey(var descracc : filind;var num_art : posint;
                 var article : artind; var succes : boolean);
                 extern;

{ Procédure d'écriture d'un article "index" }
procedure putkey(var descracc : filind;var num_art : posint;
                 var article : artind; var succes : boolean);
                 extern;

{ Cette fonction compare la première zone à la seconde; les
   deux zones sont considérées comme des tableaux de caractères
   ainsi donc des zones entières ne peuvent être comparées que
   si leur contenu est positif. La comparaison s'effectue de
   gauche à droite.
   La comparaison de zones de type réel n'est pas prévue }
function compare (var zone1 : key; var adr_z1 :posint;
                  var zone2 : key; var adr_z2 : posint;
                  var longueur : nposint;
                  var  type_cle : typezone):typetest;
                  extern;

{ Cette procédure se positionne devant le premier article
   "index" de valeur de clé supérieure à la clé donnée. }
procedure ajouter(var descracc : filind; var cle_inser : artind;
                  var succes : boolean);

var
```

```
        { Indice inférieur du sous-tableau de recherche }
        u : posint;
        { Indice supérieur du sous-tableau de recherche }
        v : posint;
        { Indice de l'élément servant à la comparaison de la
          recherche dichotomique }
        m : posint;
        { Indice de l'élément cherché }
        trouve : posint;
        { Valeur de l'élément cherché }
        cle_trouvee : artind;
        { Zone interne à la procédure permettant d'effectuer des
          comparaisons sur des zones de longueur variable }
        cle_inter : artind;
        getkey_ok,putkey_ok : boolean;
        dec_ok,rec_ok : boolean;
        UN : posint;
        comp : typetest;
        CLEF : artind;
        i : posint;

{ Procédure de décalage }
procedure decaler(var decal_ok : boolean);
begin
decal_ok := true;
{ Décalage }
while  (cle_inter.flagsup <> feof)
do begin
   CLEF.flagsup := cle_inter.flagsup;
   CLEF.num_art := cle_inter.num_art;
   CLEF.cle := cle_inter.cle;
   trouve := trouve + 1;
   getkey(descracc,trouve,cle_inter,getkey_ok);
   if getkey_ok
   then begin
        putkey(descracc,trouve,CLEF,putkey_ok);
        if not putkey_ok
        then begin
             decal_ok := false;
             cle_inter.flagsup := feof;
             end;
        end
   else begin
        decal_ok := false;
        end;
   end;
{ Ecriture de l'article de fin de fichier }
trouve := trouve + 1;
putkey(descracc,trouve,cle_inter,putkey_ok);
if not putkey_ok
then decal_ok := false;
end;

{ Procédure de recherche }
procedure recherch(var rech_ok : boolean);
```

```
var
        i : posint;

begin
rech_ok := true;
u := 1;
v := descracc.nbart + 1;
trouve := v;
while u <= v
do begin
   m := (u + v) div 2;
   getkey(descracc,m,cle_inter,getkey_ok);
   if getkey_ok
   then begin
        for i := descracc.longcle + 1 to lmaxcle
        do begin
           cle_inter.cle[i] := ' ';
           CLEF.cle[i] := ' ';
           end;
        comp := compare(cle_inter.cle,UN,CLEF.cle,UN,
                        descracc.longcle,descracc.type_cle);
        case comp
        of lower : begin
                   u := m + 1;
                   end;
           equal : begin
                   if descracc.ac_ident
                   then begin
                        rech_ok := false;
                        u := v + 1;
                        end
                   else u := m + 1;
                   end;
           greater : begin
                     trouve := m;
                     cle_trouvee.flagsup := cle_inter.flagsup;
                     cle_trouvee.num_art := cle_inter.num_art;
                     cle_trouvee.cle := cle_inter.cle;
                     v := m - 1;
                     end;
           end;
        end
   else begin
        rech_ok := false;
        u := v + 1;
        end;
   end;
cle_inter.flagsup := cle_trouvee.flagsup;
cle_inter.num_art := cle_trouvee.num_art;
cle_inter.cle := cle_trouvee.cle;
end;

{ Début de la procédure d'ajoût d'une clé }
begin
if (descracc.fd > -1)
```

```
{ Le fichier index est ouvert, donc recherche permise }
then begin
     UN := 1;
     succes := false;
     for i := 1 to descracc.longcle
     do begin
        CLEF.cle[i] := cle_inser.cle[i];
        end;
     CLEF.flagsup := cle_inser.flagsup;
     CLEF.num_art := cle_inser.num_art;
     { Recherche de l'emplacement de la nouvelle clé }
     recherch(rec_ok);
     if rec_ok
     then begin
          { Ecriture de la nouvelle clé }
          putkey(descracc,trouve,cle_inser,putkey_ok);
          if putkey_ok
          then begin
               { Décalage des autres clés }
               decaler(dec_ok);
               if dec_ok
               then succes := true;
               end;
          end;
     end
{ Le fichier index est fermé }
else succes := false;
end;
```

Closeac.p

```
#
{$c+}
const

#include "/mnt/ps3/petitpas/src/access/acces.const"

type

#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure closeac(var fdon:filcont;var find:filind;
                  var succes:integer);
{ Cette procèdure ferme un accès }

var
        ol:ok;

{ liste des fonctions utilisées }
procedure putfind(var f:filind);extern;


function close(fd:fdok):ok;extern;



{ dèbut de la procèdure CLOSEAC }
begin
        if (fdon.fd < 0) or (find.fd < 0)
        then
        succes:=2
        else
        begin

        { écriture du descripteur d'accès }
        putfind(find);
        { fermeture du fichier }
        ol:=close(find.fd);
        { test succès }
        if ol = 0
        then
                succes:=0
        else
                succes:=1;
        end;
end;
```

Closef.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure closef(var f:filcont;var succes:integer);
{ Cette procédure recopie le descripteur de fichier
   et ferme le fichier données }


var
        o1:ok;

{ liste des procédures utilisées }
function close(f:fdok):ok;extern;

procedure putfdon(var f:filcont);extern;


{ début de la procédure CLOSEF }
begin
        if f.fd < 0
        then
                succes:=2
        else
        begin
        { recopie du descripteur dans le fichier }
        putfdon(f);

        { fermeture du fichier }
        o1:=close(f.fd);
        if o1 = 0
        then
                succes:=0
        else
                succes:=1;
        end;
end;
```

Compact.p

```
#
{$c+}
procedure compact(var nom :string;var taux : integer;var result : integer;
                        var tty : text);
{ Cette  procédure compacte un fichier et ses index }

label 98;

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

var
        { descripteur de fichier ancien et nouveau }
        desfo,desfn:filcont;
        desind:filind;
        succes,i,ind,indcour,zero:integer;
        suc:boolean;
        artdon:artcont;
        ol:ok;

{ liste des fonctions utilisées }
function unlink(p:string):ok;extern;

function link(p1,p2:string):ok;extern;

function close(fd:fdok):ok;extern;

function open(n:string;r:openmode):fdok;extern;

procedure createf(var des:filcont;var succes:integer);extern;

procedure openf(var des:filcont;var succes:integer);extern;

procedure closef(var des:filcont;var suces:integer);extern;


function gen(nom:string;i:integer):string;extern;

procedure getdata(var des:filcont;var n:integer;var buf:artcont;
        var suc:boolean);extern;

procedure putdata(var des:filcont;var n:integer;var buf:artcont;
        var suc:boolean);extern;

procedure createac(var f:filcont;var fi:filind;var s:integer);extern;
```

```
procedure deleteac(var f:filcont;var fi:filind;var s:integer);extern;

procedure openac(var des:filcont;var desi:filind;var suc:integer);
        extern;

procedure closeac(var des:filcont;var desi:filind;
        var suc:integer);extern;

procedure getkey(var des:filind;var n:integer;var art:artind;
        var suc:boolean);extern;

procedure putkey(var des:filind;var n:integer;var art:artind;
        var suc:boolean);extern;

function compare(var z1:key;var p1:integer;var z2:key;var p2:integer;
                var long:nposint;var type_cle:typezone):typetest;
                extern;

procedure searchfirst(var desind:filind;var cle:key; var borne_inf:integer;
        var borne_sup:integer;var num_data : integer;var num:integer;
        var suc :boolean);extern;


procedure majind(var desfo:filcont;var desind:filind;var art :artcont;
        indcour:integer;ind:integer);
{ indcour = position du record dans l'ancien fichier
   ind = position du record dans le nouveau fichier
}


{ Cette procédure effectue la mise-à-jour d'un article index }

var
        j,i,succes:integer;
        comp:typetest;
        un ,num_data : integer;
        suc:boolean;
        artcle:artind;
        cle:key;
        num,numb:integer;


{ début de MAJIND }
begin
        { ouverture de l'accès }
        openac(desfo,desind,succes);

        un :=1;
        j:=1;
        { recherche de l'article index }
        for i:=desind.poscle to desind.longcle
        do
        begin
                cle[j]:=art.donnee[i];
                j:=j+1;
```

```
        end;
        for j:=j to lmaxcle
        do
                cle[j]:=' ';

        numb:=desind.nbart + 1;
        searchfirst(desind,cle,un,numb,num_data,num,suc);
        if not suc
        then
        begin
                writeln(tty,'MAJIND recherche sur la cle ',cle,' non reussi
                result := 3;
        end
        else
        begin
        { boucle de recherche pour trouver le bon article cle }
        repeat
                getkey(desind,num,artcle,suc);
                num:=num+1;
                comp:=compare(artcle.cle,un,cle,un,desind.longcle,
                        desind.type_cle);
        until (artcle.num_art = indcour)
                or (comp <> equal);


        if (comp <> equal)
        then
        begin
                writeln(tty,'MAJIND recherche sur la cle ',cle,' non reussi
                result := 3;
        end
        else
        begin
                { modification de l'adresse }
                artcle.num_art:=ind;
                { réécriture }
                num:=num-1;
                putkey(desind,num,artcle,suc);
        end;

        end;
        { fermeture du fichier }
        closeac(desfo,desind,succes);
end;

procedure compac(var fdon:filcont;var find:filind);

{ Cette procédure compacte un fichier index }
var
        succes:integer;


{ début de la procédure COMPAC }
begin
        { ouverture de l'accès }
```

```
        openac(fdon,find,succes);

        if (fdon.header.nbart <> find.nbart )
        then
        begin
        writeln(tty,'Pour le fichier ',fdon.nom,'et l''acces  ',find.poscle
        writeln(tty,'  ',find.longcle,' il existe une discordance sur le no
                'd''article');
        result := 2;
        end;

        { suppression de l'acces }
        deleteac(fdon,find,succes);

        { recreation de l'acces }
        createac(fdon,find,succes);
        if succes <> 0
        then
        begin
                writeln(tty,'CREATION NON REUSSIE ',succes);
                result := 3;
        end
        else

                { fermeture du fichier }
                closeac(fdon,find,succes);

end;


{ debut de COMPACT }
begin
        result := 0;
        { ouverture de l'ancien fichier }
        desfo.nom:=nom;
        openf(desfo,succes);
        if succes <> 0
        then
        begin
                writeln(tty,'COMPACT:ERREUR POUR L OUVERTURE',desfo.nom);
                writeln(tty,'SUCCES',succes);
                result:=2;
                goto 98;
        end;
        { est-il nécessaire de compacter le fichier }
        if ((desfo.header.artsup / desfo.header.nbart) < (taux/100))
        then
        begin
                result := 1;
                goto 98;
        end;

        { compactage des index de ce fichier données}
        for i:=1 to maxacc
```

```
do
        if (desfo.header.lsacces[i].longcle > 0)
        and (desfo.header.lsacces[i].poscle > 0 )
        then
        begin
                desind.poscle:=desfo.header.lsacces[i].poscle;
                desind.longcle:=desfo.header.lsacces[i].longcle;
                { compactage d'un index }
                compac(desfo,desind);
        end;

{ crêation d'un nouveau fichier }
desfn.header.lart:=desfo.header.lart;
desfn.nom:=".tempcomp";
createf(desfn,succes);
if succes <> 0
then
begin
        writeln(tty,'COMPACT:ERREUR DANS CREATION',desfn.nom);
        result := 2;
        goto 98;
end;
{ lecture du premier article ancien }
zero:=0;
ind:=1;
indcour:=1;
getdata(desfo,indcour,artdon,suc);
{ boucle de recopie }
while (artdon.flagsup <> feof)
do
begin
        if (artdon.flagsup <> fsup)
        then
        begin
                { recopie de l'article }
                putdata(desfn,ind,artdon,suc);
                { m-a-j des index }
                for i:= 1 to maxacc
                do
                if (desfo.header.lsacces[i].longcle > 0)
                and (desfo.header.lsacces[i].poscle > 0)
                then
                begin
                        desind.poscle:=desfo.header.lsacces[i].poscl
                        desind.longcle:=desfo.header.lsacces[i].long
                        majind(desfo,desind,artdon,indcour,ind);
                end;
                ind:=ind+1;
        end;
        indcour:=indcour+1;
        getdata(desfo,zero,artdon,suc);
end;
if (desfo.header.nbart >= indcour)
then
begin
```

```
            writeln(tty,'Pour le fichier ',desfo.nom,' le nombre d''article ');
            writeln(tty,' trouve est ',indcour -1,' le nombre prevu etait ',
                    desfo.header.nbart);
            result := 2;
            end;


            { ecriture de l'article de fin de fichier }
            putdata(desfn,ind,artdon,suc);
            desfn.header:=desfo.header;
            desfn.header.artsup:= 0;
            desfn.header.nbart:=ind-1;

            { fermeture du nouveau fichier }
            closef(desfn,succes);
            { fermeture de l'ancien }
            closef(desfo,succes);

            { move du nouveau fichier }
            ol:=unlink(desfo.nom);
            ol:=link(desfn.nom,desfo.nom);
            ol:=unlink(desfn.nom);

        98:
        end;
```

Compare.p

```
#

{
*******************************************************
*                                                     *
* FONCTION DE COMPARAISON DE DEUX ZONES DE MEME TYPE  *
*                    "COMPARE"                        *
*                                                     *
*******************************************************
}

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Fonction de comparaison de deux entiers }
function test_int(var z1 : key; var z2 : key) : typetest;
                  extern;

{ Cette fonction compare la première zone à la seconde; les
  deux zones doivent être de même type.
  Les types actuellement permis sont :
        - variable simple entière
        - tableau de caractères
  La comparaison de zones de type réel n'est pas prévue }
function compare (var zone1 : key; var adr_z1 :posint;
                  var zone2 : key; var adr_z2 : posint;
                  var longueur : posint;
                  var  type_cle : typezone):typetest;
var
        i : posint;
        cle1, cle2 : key;

begin
case type_cle
of C : begin
       { Tableau de caractères  }
       for i := 1 to longueur
       do begin
          cle1[i] := zone1[adr_z1 + i - 1];
          cle2[i] := zone2[adr_z2 + i - 1];
          end;
       for i := longueur + 1 to lmaxcle
       do begin
```

```
            cle1[i] := ' ';
            cle2[i] := ' ';
            end;
         if cle1 < cle2
         then compare := lower
         else if cle1 = cle2
              then compare := equal
              else compare := greater;
         end;
      E : begin
         compare := test_int(zone1,zone2);
         end;
      end;
   end;
```

Consult.p

```
#

{
*******************************************************
*                                                     *
* PRIMITIVE DE LECTURE D'ARTICLE PAR CLE : "CONSULT"  *
*                                                     *
*******************************************************
}

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Procédure de lecture d'un article "donnée" }
procedure getdata(var descrcont : filcont;var num_art : posint;
                  var article : artcont; var succes : boolean);
                  extern;

{ Procédure de recherche du premier article "donnée"
   de clé donnée }
procedure searchfirst(var descracc:filind; var clef : key;
                      var borne_inf : posint; var borne_sup : posint;
                      var n_data : posint; var n_key : posint;
                      var rech_ok :boolean);
                      extern;

{ Cette fonction compare la première zone à la seconde; les
   deux zones sont considérées comme des tableaux de caractères
   ainsi donc des zones entières ne peuvent être comparées que
   si leur contenu est positif. La comparaison s'effectue de
   gauche à droite.
   La comparaison de zones de type réel n'est pas prévue }
function compare (var zone1 : key; var adr_z1 :posint;
                  var zone2 : key; var adr_z2 : posint;
                  var longueur : nposint;
                  var  type_cle : typezone):typetest;
                  extern;

{ Procédure de lecture d'un article "index" }
procedure getkey(var descracc : filind;var num_art : posint;
                  var article : artind; var succes : boolean);
                  extern;
```

```
{ Procèdure de lecture d'un article "donnée" de clè donnée }
procedure consult (var descrcont : filcont; var descracc : filind;
                   var clef : key; var artlu : data;
                   var succes : boolean);

var
        art_inter : artcont;
       artcle : artind;
        no_data : posint;
        no_key : posint;
        get_ok,search_ok : boolean;
        i ,j: posint;
        UN ,LEC_SEQ : posint;
        comp : typetest;

begin
if (descrcont.fd = -1) or (descracc.fd = -1)
{ Le fichier données et/ou le fichier index pas ouvert(s) }
then begin
     succes := false;
     end
else begin
     UN := 1;
     LEC_SEQ := 0;
     { Accès au premier article correspondant à la clè donnée }
     searchfirst(descracc,clef,UN,descracc.nbart,no_data,no_key,
                 search_ok);
     if search_ok
     { Recherche fructueuse }
     then begin
          getdata(descrcont,no_data,art_inter,get_ok);
          if get_ok
          then begin
               if (art_inter.flagsup <> feof)
                  and (art_inter.flagsup <> fsup)
               then begin
                    for i := 1 to descrcont.header.lart
                    do artlu[i] := art_inter.donnee[i];
                    succes := true;
                    end
               else
               begin
                   if (art_inter.flagsup = feof) or
                   (( art_inter.flagsup = fsup) and (descracc.ac_ident))
                    then
                        succes:=false
                    else
             begin
                 { il existe des synonimes et le premier est supprimè }
                 { on recherche le premier synonime non supprimè }
                 { s'il existe }
                 repeat
                         getkey(descracc,LEC_SEQ,artcle,get_ok);
                         comp:=compare(artcle.cle,UN,clef,UN,
                                 descracc.longcle,descracc.type_cle);
```

```
                until (artcle.flagsup = fpres) or (comp = greater);
                if (comp = greater)
                then
                        { le synonime non supprimé n'existe pas }
                        succes:=false
                else
                begin
                   { on a trouvé un synonime }
                    getdata(descrcon,artcle.num_art,art_inter,get_ok);
                    for i := 1 to descrcont.header.lart
                    do artlu[i] := art_inter.donnee[i];
                    succes := true;
                end
              end;
         end;
                end
           else succes := false;
           end
     { Recherche infructueuse }
     else succes := false;
     end;
end;
```

Createac.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure createac(var fdon:filcont;var find:filind;
                   var succes:integer);
{ Cette procédure crée un accès et rempli
  le descripteur de cet accès }

label 9;

var
        trouve:boolean;
        i:integer;
        stdprotect:set of modebits;
        j:integer;
        ol:ok;

{ liste des fonctions utilisées }
function creat(f:string;
        s:set of modebits):fdok;extern;

function close(fd:fdok):ok;extern;

function open(f:string;
        r:openmode):fdok;extern;

procedure putfind(var f:filind);extern;


procedure creatin(var fdon:filcont;var find:filind);extern;

{ GEN génère un nom de fcihier index à partir du nom
   du fichier données et de l'indice dans la table des accès
}
function gen(nom:string;i:integer):string;extern;


{ début de la procédure CREATEAC  }
begin
        { fichier données ouvert ? }
        if (fdon.fd < 0)
        then
```

```
begin
        succes:=1;
        goto 9;
end;

{ test sur la longueur de la clè }
if (find.longcle < 0) or (find.longcle > lmaxcle)
then
begin
        succes:=2;
        goto 9;
end;

{ test sur la position de la clè}
if (find.poscle < 0) or (find.longcle > fdon.header.lart)
then
begin
        succes:=3;
        goto 9;
end;
{ test sur la position et la longueur de la clè }
if (( find.poscle + find.longcle -1 ) > fdon.header.lart )
then
begin
        succes:=4;
        goto 9;
end;

{ l'accès existe dèjà }
for i := 1 to maxacc
do
        if (find.poscle = fdon.header.lsacces[i].poscle)
        and (find.longcle = fdon.header.lsacces[i].longcle)
        then
        begin
                succes:=7;
                goto 9;
        end;

{ est-il encore possible de crèer un accès }
with fdon.header do
begin

        trouve:=false;
        for j:=1 to maxacc
        do
        if (lsacces[j].poscle = -1)
        then
        begin
                i:=j;
                j:=maxacc+1;
                trouve:=true;
        end;

        { test succès }
```

```
                if (not trouve)
                then
                begin
                        succes:=5;
                        goto 9;
                end;
                lsacces[i].poscle:=find.poscle;
                lsacces[i].longcle:=find.longcle;
                lsacces[i].type_cle:=find.type_cle;
        end;

        { création de l'accès }
        with find do
        begin
                { création du fichier d'index }
                stdprotect:=[whim,rhim,wyou,ryou,wme,rme];
                fd:=creat(gen(fdon.nom,i),stdprotect);

                { test succès }
                if (fd < 0)
                then
                begin
                        succes:=6;
                        fdon.header.lsacces[i].poscle:= -1;
                        fdon.header.lsacces[i].longcle:= -1;
                end
                else
                begin
                        o1:=close(fd);
                        { ouverture du fichier }
                        fd:=open(gen(fdon.nom,i),rw);
                        lart:=longcle+laiconst;
                        { calcul de la longueur
                        du descripteur }
                        ldesind:=longcle+ldiconst;
                        { remplissage du descripteur }
                        nbart:= 0;

                        if (find.poscle > 0)
                        and (find.longcle > 0)
                        then
                                { création des articles index }
                                creatin(fdon,find);
                        { écriture du descripteur }
                        putfind(find);

                        succes:=0;
                end;
        end;
9:
end;
```

Createf.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure createf(var f:filcont;var succes:integer);
{ Cette procèdure crée un fichier donnèes ,l'ouvre
   et rempli le descripteur de fichier }

label 9;


var
        { mode de crèation du fichier }
        stdprotect : set of modebits;
        { buffer donnèes }
        buf:artcont;
        ol:ok;
        i:integer;
        suc:boolean;
        n:posint;

{ procèdures et fonctions utilisèes }
function creat(f:string;
        s:set of modebits):fdok;extern;

function open(f:string;
        r:openmode):fdok;extern;

function close(fd:fdok):ok;extern;

procedure putdata(var f:filcont;var n:posint;var b:artcont;
        var s:boolean);extern;

procedure posfdon(var f:filcont);extern;

procedure crdesfic(var f:filcont);
{ Cette procedure crée un descripteur de fichier
  et ècrit ce descripteur sur le fichier }

var
        i:integer;

procedure putfdon(var f:filcont);extern;
```

```
begin
        with f.header do
        begin
        nbart:=0;
        artsup:=0;
        for i:= 1 to maxacc do
        begin
        { -1 indique une position vide }
                lsacces[i].poscle:= -1;
                lsacces[i].longcle:= -1;
        end;
        { calcul de la longueur du descripteur }
        ldesdon:= ldconst+maxacc*ldlconst;
        { écriture du descripteur }
        putfdon(f);
        end;
end;


{ début de la procédure CREATEF }
begin
        { test sur la longueur de l'article }
        if ((f.header.lart > lmaxart ) or (f.header.lart <= 0))
        then
        begin
                succes:=1;
                goto 9;
        end;
        { création du fichier }
        stdprotect:=[whim,rhim,wyou,ryou,wme,rme];
        f.fd:=creat(f.nom,stdprotect);

        { test succès }
        if f.fd >= 0
        then
        begin
                succes:=0;
                o1:=close(f.fd);
                { ouverture du fichier }
                f.fd:=open(f.nom,rw);
                { création du descripteur de fichier }
                crdesfic(f);
                { écriture d'un article de fin de fichier }
                buf.flagsup:=feof;
                for i:=1 to f.header.lart
                do
                        buf.donnee[i]:=chr(511);
                n:=1;
                putdata(f,n,buf,suc);
                { retour au premier article }
                posfdon(f);
        end
        else
                succes:=2;
9:
```

```
end;
```

Creatin.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure creatin(var fdon:filcont;var find:filind);
{ Cette procédure crée un fichier index }

var
        suc:boolean;
        pt:integer;
        { buffer article données }
        buf:artcont;
        { buffer article index }
        bufind:artind;
        i,j:integer;
        n,ind,zero:posint;
        hv : char;

{ liste des fonctions utilisées }

procedure getdata(var des:filcont;var n:posint;var b:artcont;
        var s:boolean);extern;

procedure putkey(var des:filind;var n:posint;var b:artind;
        var s:boolean);extern;

procedure posfind(var f:filind);extern;


procedure posfdon(var f:filcont);extern;

procedure ajouter(var des:filind;var b:artind;var s:boolean);
        extern;



{ début de la procédure CREATIN }
begin
        { écriture d'un article fin de fichier }
        posfind(find);
        bufind.flagsup:= feof;
        case find.type_cle
        of C :  begin
```

```
        hv:=chr(255);
        for j:=1 to (find.lart - 4)
        do
                bufind.cle[j]:= hv;
        end;
   E  : begin
        bufind.cle[1] := chr(255);
        bufind.cle[2] := chr(127);
        end;
end;

n:=1;
putkey(find,n,bufind,suc);

{ positionnement au premier article }
posfdon(fdon);
ind:=0;
zero:=0;
{ lecture du premier article }
getdata(fdon,ind,buf,suc);

ind:=ind+1;
{ boucle de lecture des articles du fichier donnees }
while (buf.flagsup <> feof)
do
begin
{ l'article lu est-il supprimé ? }
if (buf.flagsup <> fsup)
then
begin
        { création d'un article index }

        { extraction de la clé }
        pt:= find.poscle;
        for i:=1 to find.longcle
        do
        begin
                bufind.cle[i]:= buf.donnee[pt];
                pt:=pt+1;
        end;

        { positionnement du flag de suppression }
        bufind.flagsup:= 'p';
        { remplissage de l'adresse de l'article lu }
        bufind.num_art:=ind;
        { insertion de l'article index crée }
        ajouter(find,bufind,suc);
        find.nbart:=find.nbart+1;


end;
{ lecture de l'article donnees suivant }
ind:=ind+1;
getdata(fdon,zero,buf,suc);
end;
```

```
end;
```

Delete.p

```
#

{
************************************************
*                                              *
* PRIMITIVE DE SUPPRESSION D'ARTICLE : "DELETE" *
*                                              *
************************************************
}

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Procédure d'ouverture d'accès }
procedure openac(var descrcont : filcont; var descracc : filind;
                 var succes : integer);extern;

{ Procédure de fermeture d'accès }
procedure closeac(var descrcont : filcont; var descracc : filind;
                  var succes : integer);extern;

{ Procédure de lecture d'un article "index" }
procedure getkey(var descracc : filind;var num_art : posint;
                 var article : artind; var succes : boolean);
                 extern;

{ Procédure d'écriture d'un article "index" }
procedure putkey(var descracc : filind;var num_art : posint;
                 var article : artind; var succes : boolean);
                 extern;

{ Procédure de lecture d'un article "donnée" }
procedure getdata(var descrcont : filcont;var num_art : posint;
                 var article : artcont; var succes : boolean);
                 extern;

{ Procédure d'écriture d'un article "donnée" }
procedure putdata(var descrcont : filcont;var num_art : posint;
                 var article : artcont; var succes : boolean);
                 extern;

{ Fonction de comparaison de deux zones de longueur variable }
function compare(var z1 : key; var num_z1 : posint;
```

```
                    var z2 : key; var num_z2 : posint;
                    var longueur : nposint;
                    var type_cle : typezone) : typetest;
                    extern;

{ Procédure de recherche d'un  article "donnée" de clé donnée }
procedure searchfirst(var descracc : filind; var cle : key;
                      var borne_inf : posint; var borne_sup : posint;
                      var numdata : posint; var numkey : posint;
                      var rech_ok:boolean);
                      extern;

{ Procédure de suppression d'un article "donnée" à partir
   d'une clé identifiante }
procedure delete (var descrcont : filcont; var descracc : filind;
                  var cle : key; var succes : boolean);

var
        i : integer;
        open_ok,close_ok : integer;
        rech_ok, get_ok, put_ok : boolean;
        numdata,numkey : posint;
        art_inter : artcont;
        index : artind;
        acces : filind;
        cle_inter : key;
        no_key : posint;
        UN : posint;
        comp : typetest;

procedure supprimer;

var
        j : posint;
begin
if (descracc.poscle = descrcont.header.lsacces[i].poscle)
    and (descracc.longcle = descrcont.header.lsacces[i].longcle)
then begin
     getkey(descracc, numkey, index, get_ok);
     if get_ok
     then begin
          index.flagsup := fsup;
          putkey(descracc, numkey, index, put_ok);
          if not put_ok
          then succes := false;
          end
     else succes := false;
     end
else begin
     acces.poscle :=
           descrcont.header.lsacces[i].poscle;
     acces.longcle :=
           descrcont.header.lsacces[i].longcle;
     openac(descrcont,acces,open_ok);
     if (open_ok = 0)
```

```
     then begin
          for j := acces.poscle to acces.longcle
          do cle_inter[j - acces.poscle + 1]
                := art_inter.donnee[j];
          for j := j + 1 to lmaxcle
          do cle_inter[j] := ' ';
          searchfirst(acces, cle_inter, UN,
                      acces.nbart, numdata, no_key, rech_ok);
          if not rech_ok
          then succes := false
          else begin
               repeat getkey(acces, no_key, index, get_ok);
                      no_key := no_key + 1;
                      comp := compare(index.cle, UN,
                                      cle_inter, UN,
                                      acces.longcle,
                                      acces.type_cle);
               until (comp <> equal) or (numdata = index.num_art);
               if (comp <> equal)
               then succes := false
               else begin
                    index.flagsup := fsup;
                    no_key := no_key - 1;
                    putkey(acces, no_key, index, put_ok);
                    if not put_ok
                    then succes := false;
                    closeac(descrcont,acces,close_ok);
                    if not (close_ok = 0)
                    then succes := false;
                    end;
               end;
          end
     else succes := false;
     end;
end;

begin
if (descrcont.fd = -1) or (descracc.fd = -1)
then succes := false
else begin
     UN := 1;
     { Initialisation générale de l'indicateur de succès }
     succes := true;
     { La suppression ne peut se faire que si l'accès est
        identifiant }
     if (descracc.ac_ident)
     then begin
          searchfirst(descracc,cle,UN,descracc.nbart,numdata,
                      numkey,rech_ok);
          if rech_ok
          then begin
               getdata(descrcont,numdata,art_inter,get_ok);
               if get_ok
               then begin
                    art_inter.flagsup := fsup;
```

```
                    putdata(descrcont,numdata,art_inter,put_ok);
                    if put_ok
                    then begin
                         descrcont.header.artsup
                            := descrcont.header.artsup + 1;
                         for i := 1 to maxacc
                         do begin
                            if (descrcont.header.lsacces[i].poscle > 0)
                           and (descrcont.header.lsacces[i].longcle > 0)
                            then begin
                                 supprimer;
                                 end;
                            end
                         else succes := false;
                         end
                    else succes := false;
                    end
               else succes := false;
               end
          else succes := false;
          end;
end;
```

Deleteac.p

```
#

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Fonction simulant l'appel système UNLINK }
function unlink(p:string):ok;extern;

{ Fonction de génération d'un nom de fichier d'accès }
function gen(n:string;i:integer):string;extern;

{ Proc'dure de suppression d'un accès }
procedure deleteac(var descrcont:filcont;var descracc:filind;
                   var succes:integer);

var
        i:integer;

begin
with descrcont.header
do begin
   i:=1;
   succes:=1;
   while i<= maxacc
   do begin
      if (lsacces[i].poscle <> -1)
      and (lsacces[i].longcle <> -1)
      { traitement d'un accès effectif }
      then begin
           if (lsacces[i].poscle = descracc.poscle)
           and (lsacces[i].longcle = descracc.longcle)
           { accès à supprimer trouvé }
           then begin
                if (unlink(gen(descrcont.nom,i)) = 0)
                { suppression effectuée }
                then begin
                     lsacces[i].poscle := -1;
                     lsacces[i].longcle := -1;
                     succes:=0;
                     i:=maxacc;
                     end;
                end;
           end;
```

```
        i:=i+1;
        end;
    end;
end;
```

Gen.p

```
{$c+}
function gen(nom:string;i:integer):string;
{ Cette fonction génère un nom de fichier index
  à partir du nom de fichier données et de l'indice
   dans la table des accès
}

const
        lstring = 72;
        point = '.';
        chiffres = '0123456789';
type
        buffer=packed array[1..lstring] of char;
var
        b:buffer;
        l,ls:integer;
        tabchif:packed array [1..10] of char;

function strlen(s:string):integer;extern;
function strtobuf(nom:string;var b:buffer;l:integer):integer;
        extern;
function strbuf(var b:buffer):string;extern;
{ debut de la fonction GEN }
begin
        tabchif:=chiffres;
        ls:=strlen(nom);
        l:=strtobuf(nom,b,ls);
        if (i > 99)
        then
                b[l+1]:=chr(0)
        else
        begin
                b[l+1]:=point;
                if (i >= 10)
                then
                begin
                        b[l+2]:=tabchif[2];
                        i:=i div 10;
                end
                else
                        b[l+2]:=tabchif[1];
                b[l+3]:=tabchif[i+1];
                b[l+4]:=chr(0);
                gen:=strbuf(b);
        end;
end;
```

Getdata.p

```
#

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Fonction simulant l'appel au système "SEEK" }
function seek(f:fdok;offset:integer;r:seekreq):ok;extern;

{ Fonction simulant l'appel au système "READ" }
function uread(f:fdok;var b:artcont;long:posint):integer;extern;

{ Procédure de lecture d'un article "donnée" }
procedure getdata(var descrcont : filcont;var num_art : posint;
                  var article : artcont; var succes : boolean);

var
        { Zone de travail pour le calcul du positionnement dans
          le fichier }
        inter : integer;
        { "deplbloc" est le décalage par rapport à l'origine
          du fichier exprimé en blocs }
        deplbloc : integer;
        { "deplcar" est le décalage par rapport au bloc courant
          dans le fichier exprimé en octets }
        deplcar : integer;
        { Booléens relatifs au positionnement et à la lecture dans
          le fichier }
        pos_ok,lir_ok : boolean;

{ Procédure de lecture physique d'un article "donnée" }
procedure lire(var lire_ok : boolean);
begin
if (uread(descrcont.fd,article,descrcont.header.lart + 2)
                  = descrcont.header.lart + 2)
{ Lecture réussie }
then lire_ok := true
else lire_ok := false;
end;

{ Procédure de positionnement physique dans le fichier "données" }
procedure position(var posit_ok : boolean);
begin
posit_ok := false;
```

```
{ CETTE PROCEDURE N'ACCEPTE PAS DE TRAITER DES FICHIERS DE PLUS DE
  32767 CARACTERES CAR LA FONCTION STANDARD "TRUNC" NE FONCTIONNE
  PAS DANS UNE PROCEDURE EXTERNE (REFUS A LA COMPILATION }
{ Calcul de l'adresse physique de l'article }
inter := descrcont.header.ldesdon + (num_art - 1)
         * (descrcont.header.lart + 2);
{ Numéro de bloc physique }
deplbloc := (inter div lbloc);
{ Numéro de l'octet dans le bloc physique }
inter := inter - (deplbloc * lbloc);
deplcar := (inter);
{ Positionnement dans le fichier }
if (seek(descrcont.fd,deplbloc,absb) = 0)
{ Positionnement au bloc réussi }
then begin
     if (seek(descrcont.fd,deplcar,rel) = 0)
     { Positionnement à l'octet réussi }
     then posit_ok := true;
     end;
end;

begin
succes := false;
if not((descrcont.fd <= -1)
        or (num_art > descrcont.header.nbart))
{ Le fichier est ouvert }
then begin
     if (num_art = 0)
     { Cas de la lecture séquentielle,
        c-à-d. pas de positionnement explicite }
     then begin
          lire(lir_ok);
          if lir_ok
          then succes := true;
          end
     { Cas d'une lecture aléatoire, donc
        positionnement explicite obligatoire }
     else begin
          position(pos_ok);
          if pos_ok
          then begin
               lire(lir_ok);
               if lir_ok
               then succes := true;
               end
          end;
     end;
end;
```

Getfdon.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure getfdon(var f:filcont);

{ Cette procedure lit le descripteur d'acces du fichier donnees }


var
        ol:ok;
        l:integer;
        fdes:fdok;

{ fonctions utilisées }

function uread(fd:fdok;var buf:descript;long:posint):integer;
        extern;

function seek(fd:fdok;offset:integer;s:seekreq):ok;extern;

{ debut de la procedure GETFDON }
begin
                fdes:=f.fd;
                ol:=seek(f.fd,0,abs);
                { lecture du descripteur }
                l:=uread(f.fd,f.header,f.header.ldesdon);
                f.fd:=fdes;
end;
```

Getfind.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure getfind(var f:filind);

{ Cette procedure lit le descripteur d'index }


var
        ol:ok;
        l:integer;
        fdes:fdok;

{ fonctions utilisées }

function uread(fd:fdok;var buf:filind;long:posint):integer;
        extern;

function seek(fd:fdok;offset:integer;s:seekreq):ok;extern;

{ début de la procédure GETFIND }
begin
                fdes:=f.fd;
                ol:=seek(f.fd,0,abs);
                { lecture du descripteur }
                l:=uread(f.fd,f,f.ldesind);
                f.fd:=fdes;
end;
```

Getkey.p

```
#

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Fonction simulant l'appel au système "SEEK" }
function seek(f:fdok;offset:integer;r:seekreq):ok;extern;

{ Fonction simulant l'appel au système "READ" }
function uread(f:fdok;var b:artind;long:posint):integer;extern;

{ Procédure de lecture d'un article "clé" }
procedure getkey(var descracc : filind;var num_art : posint;
                 var article : artind; var succes : boolean);

var
        { Zone de travail pour le calcul du positionnement dans
          le fichier }
        inter : integer;
        { "deplbloc" est le décalage par rapport à l'origine
          du fichier exprimé en blocs }
        deplbloc : integer;
        { "deplcar" est le décalage par rapport au bloc courant
          dans le fichier exprimé en octets }
        deplcar : integer;
        { Booléens relatifs au positionnement et à la lecture dans
          le fichier }
        lir_ok,pos_ok : boolean;

{ Procédure de lecture physique d'un article "clé" }
procedure lire(var lire_ok : boolean);
begin
if (uread(descracc.fd,article,descracc.lart)
    = descracc.lart)
{ Lecture réussie }
then lire_ok := true
else lire_ok := false;
end;

{ Procédure de positionnement physique dans le fichier "index" }
procedure position(var posit_ok : boolean);
begin
posit_ok := false;
```

```
{ CETTE PROCEDURE N'ACCEPTE PAS DE TRAITER DES FICHIERS DE PLUS DE
  32767 CARACTERES CAR LA FONCTION STANDARD "TRUNC" NE FONCTIONNE
  PAS DANS UNE PROCEDURE EXTERNE (REFUS A LA COMPILATION }
{ Calcul de l'adresse physique de l'article }
inter := descracc.ldesind + (num_art - 1)
         * descracc.lart;
{ Numéro de bloc physique }
deplbloc := inter div lbloc;
{ Numéro de l'octet dans le bloc physique }
inter := inter - (deplbloc * lbloc);
deplcar := inter;
{ Positionnement dans le fichier }
if (seek(descracc.fd,deplbloc,absb) = 0)
{ Positionnement au bloc réussi }
then begin
     if (seek(descracc.fd,deplcar,rel) = 0)
     { Positionnement à l'octet réussi }
     then posit_ok := true;
     end;
end;

begin
succes := false;
if (descracc.fd > -1)
then begin
     if (num_art = 0)
     { Cas de la lecture séquentielle,
        c-à-d. pas de positionnement explicite }
     then begin
          lire(lir_ok);
          if lir_ok
          then succes := true;
          end
     { Cas d'une lecture aléatoire, donc
       positionnement explicite obligatoire }
     else begin
          position(pos_ok);
          if pos_ok
          then begin
               lire(lir_ok);
               if lir_ok
               then succes := true;
               end;
          end;
     end;
end;
```

Getpdata.p

```
#

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Fonction simulant l'appel au système "SEEK" }
function seek( f:fdok;offset:integer;r:seekreq):ok;extern;

{ Fonction simulant l'appel au système "READ" }
function uread( f:fdok;var b:artcont;long:posint):integer;extern;

{ Procédure de lecture de l'article "donnée" à manipuler }
procedure getpdata(var descrcont : filcont;
                   var article : artcont; var succes : boolean);



begin
succes := false;
if (descrcont.fd <> -1)
{ Le fichier est fermé }
then begin
     if (seek(descrcont.fd,-(descrcont.header.lart + 2),rel) = 0)
     then begin
          if (uread(descrcont.fd,article,descrcont.header.lart + 2)
                              = descrcont.header.lart + 2)
          { Lecture réussie }
          then succes := true;
          end;
     end;
end;
```

Insert.p

```
#

{
************************************************
*                                              *
* PRIMITIVE D'INSERTION D'UN ARTICLE : "INSERT" *
*                                              *
************************************************
}

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Procédure d'écriture d'un article "donnée" }
procedure putdata(var descrcont : filcont;var num_art : posint;
                  var article : artcont; var succes : boolean);
                  extern;

{ Procédure d'ouverture d'un accès }
procedure openac(var descrcont : filcont; var descracc : filind;
                  var succes : integer);extern;

{ Procédure d'ajoût d'un clé dans le fichier "index" }
procedure ajouter(var descracc : filind; var cle : artind;
                   var succes : boolean);extern;

{ Procédure fermeture d'un accès }
procedure closeac(var descrcont : filcont; var descracc : filind;
                   var succes : integer);extern;

{ Procédure ouverture d'un fichier }
procedure openf(var f:filcont;var suc:integer);extern;

{ Procédure de fermeture d'un fichier }
procedure closef(var f:filcont;var succes:integer);extern;

{ Procédure d'ajoût d'un article dans le fichier "données" }
procedure insert (var descrcont : filcont; var article : data;
                  var succes : boolean);

var
        i,j : integer;
        put_ok : boolean;
```

```
        ajout_ok : boolean;
        close_ok,open_ok:integer;
        descracc : filind;
        clef : artind;
        art_inser : artcont;
        ECR_SEQ : posint;
        ECR_SUIV : posint;

begin
succes := false;
ECR_SEQ := 0;
if (descrcont.fd > -1)
then begin
     { Tous les nombres entiers sont définis de type "posint", c-à-d.
       compris entre 0 et "maxint"; afin qu'il n'y ait pas de
       dépassement de la borne "maxint" lors d'une exécution
       limite et donc, de fin anormale du programme ne pouvant
       être contrôlée par le programme lui-même, nous imposons
       par cette condition un nombre maximum d'articles par fichier
       de "maxint - 2" }
     if (descrcont.header.nbart < maxart)
     { Il y a encore place dans le fichier }
     then begin
          art_inser.flagsup := fpres;
          for i := 1 to descrcont.header.lart
          do art_inser.donnee[i] := article[i];
          { Ecriture de l'article précédé d'une zone "flag"
            initialisée à présent c-à-d. fpres }
          ECR_SUIV := descrcont.header.nbart + 1;
          putdata(descrcont,ECR_SUIV,
                  art_inser,put_ok);
          if put_ok
          then begin
               descrcont.header.nbart
                                  := descrcont.header.nbart + 1;
               { Ecriture de l'article "donnée" de fin de fichier }
               art_inser.flagsup := feof;
               for i := 1 to descrcont.header.lart
               do art_inser.donnee[i] := ' ';
               putdata(descrcont,ECR_SEQ,art_inser,put_ok);
               if put_ok
               then begin
                    succes := true;
                    { M-à-J des fichiers "index" associés à ce
                      fichier "données" }
                    for i := 1 to maxacc
                    do begin
                       if ((descrcont.header.lsacces[i].poscle > 0)
                         and (descrcont.header.lsacces[i].longcle > 0))
                       then begin
                            descracc.poscle
                              := descrcont.header.lsacces[i].poscle;
                            descracc.longcle
                              := descrcont.header.lsacces[i].longcle;
                            openac(descrcont,descracc,open_ok);
```

```
                                        if open_ok = 0
                                        then begin
                                             { Constitution de l'article
                                               "index" à ajouter }
                                             clef.flagsup := fpres;
                                             clef.num_art
                                                 := descrcont.header.nbart;
                                             for j := 1
                                             to descrcont.header.lsacces[i].longcle
                                             do clef.cle[j] :=
                                                article[descracc.poscle + j -1];
                                             ajouter(descracc,clef,ajout_ok);
                                             if ajout_ok
                                             then begin
                                                  descracc.nbart
                                                          :=descracc.nbart+1;
                                                  closeac(descrcont,descracc,
                                                          close_ok);
                                                  if not (close_ok = 0)
                                                  then succes := false;
                                                  end
                                             else begin
                                                  succes := false;
                                                  closeac(descrcont,descracc,
                                                          close_ok);
                                                  end;
                                             end
                                        else begin
                                             succes := false;
                                             end;
                                        end;
                                   end;
                              end;
                         end;
                    end;
               end;
          end;
```

Modify.p

```
#

{
************************************************
*                                              *
* PRIMITIVE DE MODIFICATION D'ARTICLE : "MODIFY" *
*                                              *
************************************************
}

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Procédure de relecture de l'article "donnée" le dernier traité }
procedure getpdata(var descrcont : filcont;
                   var article : artcont; var succes : boolean);
                   extern;

{ Procédure de réécriture d'un article "donnée" }
procedure putpdata(var descrcont : filcont;
                   var article : artcont; var succes : boolean);
                   extern;

{ Fonction de comparaison de deux zones de longueur variable }
function compare(var z1 : data; var num_z1 : nposint;
                 var z2 : data; var num_z2 : nposint;
                 var longueur : nposint;
                 var  type_cle : typezone) : typetest;
                 extern;

{ Avant d'utiliser cette procédure, il faut faire une lecture
   de l'article par une procédure CONSULT ou READNEXT
   Cette procédure interdit toute modification de clé }
procedure modify(var descrcont : filcont; var descracc : filind;
                 var art_modif : data; var succes : boolean);

var
        i : integer;
        art_inter : artcont;
        get_ok,put_ok : boolean;
        comp : typetest;

begin
```

```
if (descrcont.fd = -1) or (descracc.fd =  -1)
{ Un des fichiers est fermé }
then begin
     succes := false;
     end
else begin
     { Lecture de l'article à modifier, pour vérifier la
       validité de la modification }
     getpdata(descrcont,art_inter,get_ok);
     if get_ok
     then begin
          succes := true;
          { Vérification pour chaque accès existant }
          for i:= 1 to maxacc
          do begin
             if (descrcont.header.lsacces[i].poscle > 0)
                and (descrcont.header.lsacces[i].longcle > 0)
             then begin
                  comp := compare(art_modif,
                                  descrcont.header.lsacces[i].poscle,
                                  art_inter.donnee,
                                  descrcont.header.lsacces[i].poscle,
                                  descrcont.header.lsacces[i].longcle,
                                  descrcont.header.lsacces[i].type_cle);
                  if (comp <> equal)
                  then begin
                       succes := false;
                       i := maxacc;
                       end;
                  end;
             end;
          if succes
          then begin
               { Ecriture de l'article modifié et contrôlé }
               art_inter.donnee := art_modif;
               putpdata(descrcont,art_inter,put_ok);
               if not put_ok
               then succes := false;
               end;
          end
     else succes := false;
     end;
end;
```

Openac.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure openac(var fdon:filcont;var find:filind;
                 var succes:integer);
{ Cette procèdure ouvre un fichier index
   et rempli le descripteur d'accès }

var
        j,i:integer;
        trouve : boolean;

{ liste des fonctions utilisées }
procedure getfind(var f:filind);extern;

function open(f:string;
        r:openmode):fdok;extern;

{ GEN g génère un nom de fichier index à partir du
   nom du fichier données et de l'indice dans la table
   des accès
}
function gen(nom:string;i:integer):string;extern;


{ début de la procédure OPENAC }
begin
        { fichier données ouvert ? }
        if (fdon.fd >= 0)
        then
        begin
        { recherche de la description de la clè
        dans le descripteur }
        trouve := false;
        for i:=1 to maxacc
        do
          if  ((fdon.header.lsacces[i].poscle  = find.poscle) and
                (fdon.header.lsacces[i].longcle  = find.longcle))
                then
                begin
                        trouve:=true;
```

```
                        j:=i;
                        i:=maxacc+1;
                end;
        { test succès }
        if trouve
        then
        begin
                { ouverture du fichier index }
                find.fd:=open(gen(fdon.nom,j),rw);
                { test succès }
                if find.fd >= 0
                then
                begin
                        { calcul de la longueur
                        du descripteur }
                        find.ldesind:=lmaxcle+ldiconst;
                        { lecture du descripteur }
                        getfind(find);
                        succes:=0;
                end
                else
                        succes:=1;
        end
        else
                succes:=2;
        end
        else
                succes:=3;
end;
```

Openf.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure openf(var f:filcont;var succes:integer);

{ Cette procédure ouvre un fichier données et
   rempli le descripteur de fichier }



{ fonctions utilisées }
function open( f:string;
        r:openmode):fdok;extern;

procedure getfdon(var f:filcont);extern;

procedure posfdon(var f:filcont);extern;

{ début de la procédure OPENF }
begin
        { ouverture du fichier }
        f.fd:= open( f.nom,rw);
        { test succès }
        if f.fd >= 0
        then
        begin
                succes:= 0;
                { calcul de la longueur du descripteur }
                f.header.ldesdon:=ldconst+maxacc * ldlconst;
                { lecture du descripteur }
                getfdon( f);
                { positionnement au premier article }
                posfdon( f);
        end
        else
                succes:=1;
end;
```

Posfdon.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

procedure posfdon(var f:filcont);
{ Cette procedure se positionne au premier article fichier donnees }

var
        ol:ok;

function seek(fd:fdok;offset:integer;s:seekreq):ok;extern;
begin
        { positionnement }
        ol:=seek(f.fd,f.header.ldesdon,abs);
end;
```

Posfind.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

procedure posfind(var f:filind);
{ Cette procedure se positionne au premier article index }

var
        ol:ok;

function seek(fd:fdok;offset:integer;s:seekreq):ok;extern;
begin
        { positionnement }
        ol:=seek(f.fd,f.ldesind,abs);
end;
```

Putdata.p

```
#

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Fonction simulant l'appel au système "SEEK" }
function seek(f:fdok;offset:integer;r:seekreq):ok;extern;

{ Fonction simulant l'appel au système "WRITE" }
function uwrite(f:fdok;var b:artcont;long:posint):ok;extern;

{ Procédure d'écriture d'un article "donnée" }
procedure putdata(var descrcont : filcont;var num_art : posint;
                  var article : artcont; var succes : boolean);

var
        { Zone de travail pour le calcul du positionnement dans
          le fichier }
        inter : integer;
        { "deplbloc" est le décalage par rapport à l'origine
          du fichier exprimé en blocs }
        deplbloc : integer;
        { "deplcar" est le décalage par rapport au bloc courant
          dans le fichier exprimé en octets }
        deplcar : integer;
        { Booléens relatifs au positionnement et à l'écriture dans
          le fichier }
        posit_ok : boolean;
        ecrir_ok : boolean;

{ Procédure de positionnement physique dans le fichier "données" }
procedure position (var pos_ok : boolean);
begin
pos_ok := false;
{ CETTE PROCEDURE N'ACCEPTE PAS DE TRAITER DES FICHIERS DE PLUS DE
  32767 CARACTERES CAR LA FONCTION STANDARD "TRUNC" NE FONCTIONNE
  PAS DANS UNE PROCEDURE EXTERNE (REFUS A LA COMPILATION }
{ Calcul de l'adresse physique de l'article }
inter := descrcont.header.ldesdon + (num_art - 1)
        * (descrcont.header.lart + 2);
{ Numéro de bloc physique }
deplbloc := (inter div lbloc);
{ Numéro de l'octet dans le bloc physique }
```

```
inter := inter - (deplbloc * lbloc);
deplcar := (inter);
{ Positionnement dans le fichier }
if (seek(descrcont.fd,deplbloc,absb) = 0)
{ Positionnement au bloc réussi }
then begin
     if (seek(descrcont.fd,deplcar,rel) = 0)
     { Positionnement à l'octet réussi }
     then pos_ok := true;
     end;
end;

{ Procédure d'écriture physique d'un article "donnée" }
procedure ecrire(var ecr_ok : boolean);
begin
if (uwrite(descrcont.fd,article,descrcont.header.lart + 2) = 0)
{ Ecriture réussie }
then ecr_ok := true
else ecr_ok := false;
end;

begin
succes := false;
if (descrcont.fd > -1)
{ Le fichier est ouvert }
then begin
     if (num_art = 0)
     { Cas de l écriture séquentielle,
        c-à-d. pas de positionnement explicite }
     then begin
          ecrire(ecrir_ok);
          if ecrir_ok
             then succes := true;
          end
     { Cas d'une écriture aléatoire, donc
        positionnement explicite obligatoire }
     else begin
          position(posit_ok);
          if posit_ok
          then begin
               ecrire(ecrir_ok);
               if ecrir_ok
               then succes := true;
               end;
          end;
     end;
end;
```

Putfdon.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure putfdon(var f:filcont);

{ Cette procedure ecrit le descripteur d'acces du fichier donnees }


var
        ol:ok;
        l:integer;

{ fonctions utilisées }

function uwrite(fd:fdok;var buf:descript;long:posint):integer;
        extern;

function seek(fd:fdok;offset:integer;s:seekreq):ok;extern;

{ début de la procédure PUTFDON }
begin
                ol:=seek(f.fd,0,abs);
                { ecriture du descripteur }
                l:=uwrite(f.fd,f.header,f.header.ldesdon);
end;
```

Putfind.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"


procedure putfind(var f:filind);

{ Cette procedure ecrit le descripteur d'index }


var
        ol:ok;
        l:integer;

{ fonctions utilisées }

function uwrite(fd:fdok;var buf:filind;long:posint):integer;
        extern;

function seek(fd:fdok;offset:integer;s:seekreq):ok;extern;

{ début de la procédure PUTFIND }
begin
                ol:=seek(f.fd,0,abs);
                { ecriture du descripteur }
                l:=uwrite(f.fd,f,f.ldesind);
end;
```

Putkey.p

```
#

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Fonction simulant l'appel au système "SEEK" }
function seek(f:fdok;offset:integer;r:seekreq):ok;extern;

{ Fonction simulant l'appel au système "WRITE" }
function uwrite(f:fdok;var b:artind;long:posint):ok;extern;

{ Prpcédure d'écriture d'un article "clé" }
procedure putkey(var descracc : filind;var num_art : posint;
                 var article : artind; var succes : boolean);

var
        { Zone de travail pour le calcul du positionnement dans
          le fichier }
        inter : integer;
        { "deplbloc" est le décalage par rapport à l'origine
          du fichier exprimé en blocs }
        deplbloc : integer;
        { "deplcar" est le décalage par rapport au bloc courant
          dans le fichier exprimé en octets }
        deplcar : integer;
        { Booléens relatifs au positionnement et à l'écriture dans
          le fichier }
        pos_ok,ecr_ok : boolean;

{ Procédure d'écriture physique d'un article "clé" }
procedure ecrire(ecrir_ok : boolean);
begin
if (uwrite(descracc.fd,article,descracc.lart) = 0)
{ Ecriture réussie }
then ecrir_ok := true
else ecrir_ok := false;
end;

{ Procédure de positionnement physique dans le fichier "index" }
procedure position(var posit_ok : boolean);
begin
posit_ok := false;
{ CETTE PROCEDURE N'ACCEPTE PAS DE TRAITER DES FICHIERS DE PLUS DE
```

```
  32767 CARACTERES CAR LA FONCTION STANDARD "TRUNC" NE FONCTIONNE
  PAS DANS UNE PROCEDURE EXTERNE (REFUS A LA COMPILATION }
{ Calcul de l'adresse physique de l'article }
inter := descracc.ldesind + (num_art - 1)
         * descracc.lart;
{ Numèro de bloc physique }
deplbloc := (inter div lbloc);
{ Numèro de l'octet dans le bloc physique }
inter := inter - (deplbloc * lbloc);
deplcar := (inter);
{ Positionnement dans le fichier }
if (seek(descracc.fd,deplbloc,absb) = 0)
{ Positionnement au bloc rèussi }
then begin
     if (seek(descracc.fd,deplcar,rel) = 0)
     { Positionnement à l'octet rèussi }
     then posit_ok := true;
     end;
end;

begin
if (descracc.fd = -1)
{ Le fichier est fermè }
then succes := false
else begin
     if (num_art = 0)
     { Cas de lècriture sèquentielle,
        c-à-d. pas de positionnement explicite }
     then begin
          ecrire(ecr_ok);
          if ecr_ok
          then succes := true
          else succes := false;
          end
     { Cas d'une ècriture alèatoire, donc
        positionnement explicite obligatoire }
     else begin
          position(pos_ok);
          if pos_ok
          then begin
               ecrire(ecr_ok);
               if ecr_ok
               then succes := true
               else succes := false;
               end
          else succes := false;
          end;
     end;
end;
```

Putpdata.p

```
#

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Fonction simulant l'appel au système "SEEK" }
function seek(f:fdok;offset:integer;r:seekreq):ok;extern;

{ Fonction simulant l'appel au système "WRITE" }
function uwrite(f:fdok;var b:artcont;long:posint):ok;extern;

{ Procédure d'écriture de l'article "donnée" à manipuler }
procedure putpdata(var descrcont : filcont;
                   var article : artcont; var succes : boolean);


begin
succes := false;
if (descrcont.fd <> -1)
{ Le fichier est fermé }
then begin
     if (seek(descrcont.fd,-(descrcont.header.lart + 2),rel) = 0)
     then begin
          if (uwrite(descrcont.fd,article,descrcont.header.lart + 2)
                              = 0)
          { Ecriture réussie }
          then succes := true;
          end;
     end;
end;
```

Readnext.p

```
#

{
*****************************************************************
*                                                               *
* PRIMITIVE DE LECTURE SEQUENTIELLE D'UN ARTICLE : "READNEXT"   *
*                                                               *
*****************************************************************
}

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Procédure de lecture d'un article "index" }
procedure getkey(var descracc : filind;var num_art : posint;
                 var article : artind; var succes : boolean);
                 extern;

{ Procédure de lecture d'un article "donnée" }
procedure getdata(var descrcont : filcont;var num_art : posint;
                  var article : artcont; var succes : boolean);
                  extern;

{ Procédure de lecture séquentielle d'un article "donnée" }
procedure readnext (var descrcont : filcont; var descracc : filind;
                    var artlu : data; var endoffile : boolean;
                    var succes : boolean);

var
        artindlu : artind;
        getk_ok,getd_ok : boolean;
        art_inter : artcont;
        i : posint;
        LEC_SEQ : posint;

begin
LEC_SEQ := 0;
if (descrcont.fd = -1) or (descracc.fd = -1)
{ Les fichiers ne sont pas ouverts, donc refus }
then begin
     succes := false;
     end
{ Les fichiers sont ouverts }
```

```
else begin
     if (descracc.poscle = 0) and (descracc.longcle = 0)
     { Accès séquentiel }
     then begin
          repeat
          getdata(descrcont,LEC_SEQ,art_inter,getd_ok);
          if getd_ok
          then begin
               if (art_inter.flagsup = feof)
               { La fin du fichier données est détectée }
               then begin
                    endoffile := true;
                    succes := false;
                    end
               else begin
                    endoffile := false;
                    succes := true;
                    for i := 1 to descrcont.header.lart
                    do artlu[i] := art_inter.donnee[i];
                    end
               end
          else begin
               succes := false;
               end;
          until (art_inter.flagsup <> fsup);
          end
     { Accès par clé }
     else begin
           repeat
          getkey(descracc,LEC_SEQ,artindlu,getk_ok);
          if getk_ok
          { La lecture a réussi }
          then begin
               if (artindlu.flagsup = feof)
               { La fin du fichier index est détectée }
               then begin
                    endoffile := true;
                    succes := false;
                    end
               else begin
                    getdata(descrcont,artindlu.num_art,
                            art_inter,getd_ok);
                    if getd_ok
                    then begin
                         succes := true;
                         endoffile := false;
                         for i := 1 to descrcont.header.lart
                         do artlu[i] := art_inter.donnee[i];
                         end
                    else succes := false;
                    end
               end
          { La lecture a échoué }
          else succes := false;
           until (artindlu.flagsup <> fsup);
```

```
            end
        end
    end;
```

Searchfirst.p

```
#

{
************************************************************
*                                                          *
*                      "SEARCHFIRST"                       *
* PROCEDURE DE RECHERCHE DU PREMIER ARTICLE CORRESPONDANT  *
* A UNE CLE DONNEE UTILISEE PAR LA PRIMITIVE "CONSULT"     *
*                                                          *
************************************************************
}

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

{ Procédure de lecture d'un article "index" }
procedure getkey(var descracc : filind;var num_art : posint;
                 var article : artind; var succes : boolean);
                 extern;

{ Fonction de comparaison de deux zones de longueur variable }
function compare(var z1 : key; var num_z1 : posint;
                 var z2 : key; var num_z2 : posint;
                 var longueur : nposint;
                 var type_cle : typezone) : typetest;
                 extern;

{ Procédure de recherche de la position du premier article
   "donnée" de clé donnée }
procedure searchfirst(var descracc : filind; var clef : key;
                      var borne_inf : posint; var borne_sup : posint;
                      var no_art : posint; var no_key : posint;
                      var s_ok : boolean);

var
        comp : typetest;
        art_ind : artind;
        indice,i : posint;
        get_ok ,posit_ok: boolean;
        UN : posint;
        b_sup,b_inf:posint;
```

```
{ Procédure de positionnement physique dans le fichier "index" }
procedure position(var posit_ok : boolean);
var
        { Zone de travail pour le calcul du positionnement dans
          le fichier }
        inter : integer;
        { "deplbloc" est le décalage par rapport à l'origine
          du fichier exprimé en blocs }
        deplbloc : integer;
        { "deplcar" est le décalage par rapport au bloc courant
          dans le fichier exprimé en octets }
        deplcar : integer;

{ Fonction simulant l'appel au système "SEEK" }
function seek(f:fdok;offset:integer;r:seekreq):ok;extern;

begin
posit_ok := false;
{ Calcul de l'adresse physique de l'article }
inter := descracc.ldesind + ((no_key + 1) - 1)
         * descracc.lart;
{ Numéro de bloc physique }
deplbloc:= inter div lbloc;
{ Numéro de l'octet dans le bloc physique }
inter := inter - (deplbloc * lbloc);
deplcar := inter;
{ Positionnement dans le fichier }
if (seek(descracc.fd,deplbloc,absb) = 0)
{ Positionnement au bloc réussi }
then begin
     if (seek(descracc.fd,deplcar,rel) = 0)
     { Positionnement à l'octet réussi }
     then posit_ok := true;
     end;
end;

{ debut de searchfirst }
begin
if (descracc.fd > -1)
{ Le fichier est ouvert }
then
begin
        UN :=1;
        b_sup:=borne_sup;b_inf:=borne_inf;
        s_ok :=false;
        { recherche d'une cle }
        while (b_inf <= b_sup )
        do
        begin
                indice:=(b_sup + b_inf) div 2;
                { lecture d'un article }
                getkey(descracc,indice,art_ind,get_ok);
                if get_ok
                then
                begin
```

```
                    for i := descracc.longcle + 1 to lmaxcle
                    do begin
                       art_ind.cle[i] := ' ';
                       end;
                    comp := compare(clef,UN,art_ind.cle,UN,
                                    descracc.longcle,
                                    descracc.type_cle);
                    case comp
                    of lower : begin
                                  b_sup :=indice -1;
                               end;
                       equal : begin
                                  { on a trouve }
                                  b_inf:=b_sup +1;
                                  s_ok:=true;
                               end;
                       greater : begin
                                    b_inf:=indice + 1;
                                 end;
                       end;
          end
          else
                  s_ok:=false;
end;

if (s_ok)
then
begin
          { preparation des resultats }
          no_key:=indice;
          no_art:=art_ind.num_art;
          { recherche de la premiere cle }
          while (indice > 1)
          do
          begin
                  indice :=indice -1;
                  getkey(descracc,indice,art_ind,get_ok);
                  if get_ok
                  then
                  begin
                           for i := descracc.longcle + 1 to lmaxcle
                           do begin
                              art_ind.cle[i] := ' ';
                              end;
                           comp := compare(clef,UN,art_ind.cle,UN,
                                           descracc.longcle,
                                           descracc.type_cle);
                           case comp
                          of lower,greater :
                                  begin
                                          { arret }
                                          indice:=0;
                                  end;
                             equal :
                                  begin
```

```
                                                    {  sauvetage }
                                                    no_key:=indice;
                                                    no_art:=art_ind.num_art;
                                                end;
                                        end;
                                end
                                else
                                        s_ok:=false;
                        end;
                        if s_ok
                        then
                        begin
                                { posionnement a l'article index suivant }
                                position(posit_ok);
                                if not posit_ok
                                then
                                        s_ok:=false;
                        end;
                end;
end
{ Le fichier n'est pas ouvert }
else s_ok := false;
end;
```

Testint.p

```
#

{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"

function test_int (var nb1 : integer; var nb2 : integer) : typetest;
begin
if nb1 < nb2
then test_int := lower
else if nb1 = nb2
     then test_int := equal
     else test_int := greater;
end;
```

PRIMITIVES DE COMMUNICATION.

Constantes.

```
{ Longueur du descripteur de boite aux lettres}
ldesb = 10;
{ Longueur maximum d'un fichier UNIX }
lfic = 1024;
```

Types.

```
{ Descripteur de boite aux lettres }
filbox = record
        { pointeur en ecriture }
        ptwrite : integer;
        { pointeur en lecture }
        ptread : integer;
        {nombre de message actif }
        nbre :integer;
        { longueur des messages }
        lmes : integer;
        { nombre maximum de messages de ce type
        pour la boite aux lettres }
        maxmes : integer;
end;
```

Creatbox.p

```
#
{$c+}

const
#include "/mnt/ps3/petitpas/src/access/acces.const"
#include "/mnt/ps3/petitpas/src/communication/box.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/communication/box.type"

function creatbox(box:string;lmes :integer):integer;
{ Cette fonction cree une boite aux lettres (box)
 dont la longueur des messages est donnee(lmes) et fixe }

var
        fd:fdok;
        { descripteur de boite }
        desbox:filbox;
        std : set of modebits;
        ol:ok;

{fonctions utilisees }
function open(p:string;m:openmode):fdok;extern;

function close(f:fdok):ok;extern;

function creat(p:string;m:set of modebits):fdok;extern;

procedure putfbox(fd:fdok;var f:filbox);extern;

{ debut de CREATBOX }
begin
        { la boite aux lettres existe-t-elle ?}
        fd:=open(box,rw);
        if (fd <> -1 )
        then
        begin
                creatbox:=1;
                ol:=close(fd);
        end
        else
        begin
                { test sur la longueur des messages }
                if ((lmes <= 0) or (lmes > lbloc))
                then
                        creatbox:=2
                else
```

```
                    begin
                            { creation de la boite }
                            std:=[xhim,whim,rhim,xyou,wyou,ryou,
                                    xme,wme,rme];
                            fd:=creat(box,std);
                            if (fd = -1 )
                            then
                                    creatbox :=3
                            else
                            begin
                            { remplissage du descripteur de boite }
                            desbox.ptread:=1;
                            desbox.ptwrite:=1;
                            desbox.lmes:=lmes;
                            desbox.nbre:=0;
                            desbox.maxmes:=(lfic - ldesb) div(lmes);
                            { ecriture du descripteur }
                            putfbox(fd,desbox);
                            { fermeture }
                            o1:=close(fd);
                            creatbox:=0;
                            end;
                    end;
            end;
    end;
```

Delbox.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
#include "/mnt/ps3/petitpas/src/communication/box.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/communication/box.type"

function delbox(box:string):integer;
{ Cette fonction supprime une boite aux lettres }

var
        ol:ok;
{fonctions utilisees }
function unlink(s:string):ok;extern;

{ debut de DELBOX }
begin
        {suppression de la boite }
        ol:=unlink(box);
        if (ol = 0)
        then
                delbox:=0
        else
                delbox:=1;
end;
```

Getfbox.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
#include "/mnt/ps3/petitpas/src/communication/box.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/communication/box.type"

procedure getfbox( fd:fdok;var desbox:filbox);
{ Cette procedure lit un descripteur de boite aux lettres }

var
        ol:ok;
        l:integer;


{fonctions utilisees }
function seek( f:fdok;offset:integer;r:seekreq):ok;extern;

function uread( f:fdok;var d:filbox;l:integer):integer;extern;

{ debut de GETFBOX }
begin
        { lecture du descripteur de boite }
        ol:=seek( fd,0,abs );
        l:=uread( fd,desbox,ldesb);
end;
```

Getmess.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
#include "/mnt/ps3/petitpas/src/communication/box.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/synchro/sema.type"
#include "/mnt/ps3/petitpas/src/communication/box.type"

procedure getmess(box:string;var s:path;var mess:bufbloc;var succes:integer);
{ Cette procedure lit le premier message disponible dans la boite }

var
        fd,fdsem :fdok;
        ver :verrou;
        o1:ok;
        desbox:filbox;

{ fonctions utilisees }
#include "/mnt/ps3/petitpas/src/synchro/sema.funct"

function open(s:string;m:openmode):fdok;extern;

function close(fd:fdok):ok;extern;

procedure getfbox(fd:fdok;var f:filbox);extern;

procedure putfbox(fd:fdok;var f:filbox);extern;

function rmess(fd:fdok;var f:filbox;var mess:bufbloc):ok;extern;

{ debut de GETMESS }
begin
        { ouverture de la boite }
        fd:=open(box,rw);
        if (fd = -1)
        then
                succes:=1
        else
        begin
                { verrouillage }
                fdsem:=semopen(s);
                if (fdsem <> -1 )
                then
                        ver:=lock(fdsem);
```

```
                {lecture du descripteur }
                getfbox( fd,desbox);
                { existe-t-il un message ? }
                if (desbox.nbre <> 0)
                then
                begin
                        { lecture du message }
                        ol:=rmess( fd,desbox,mess );
                        if (ol = -1)
                        then
                                {echec de la lecture}
                                succes:=2
                        else
                        begin
                        { mise-a-jour du nombre des messages}
                        desbox.nbre:=desbox.nbre-1;
                        { mise-a-jour du pointeur }
                        desbox.ptread:=desbox.ptread+1;
                        { fin de boite atteinte ? }
                        if (desbox.ptread > desbox.maxmes)
                        then
                                        desbox.ptread:=1;
                        { ecriture du descripteur }
                        putfbox( fd,desbox);
                        succes:=0;
                        end;
                end
                else
                        { plus de message }
                        succes:=3;
                {deverrouillage}
                if ( fdsem <> -1)
                then
                        ver:=unlock( fdsem);

                ol:=close( fdsem);
                {fermeture}
                ol:=close( fd);
        end;
end;
```

Putfbox.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
#include "/mnt/ps3/petitpas/src/communication/box.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/communication/box.type"

procedure putfbox( fd:fdok;var desbox:filbox);
{ Cette procedure ecrit un descripteur de boite aux lettres }

var
        ol:ok;
        l:integer;


{fonctions utilisees }
function seek( f:fdok;offset:integer;r:seekreq):ok;extern;

function uwrite( f:fdok;var d:filbox;l:integer):integer;extern;

{ debut de PUTFBOX }
begin
        { ecriture du descripteur de boite }
        ol:=seek( fd,0,abs );
        l:=uwrite( fd,desbox,ldesb );
end;
```

Putmess.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
#include "/mnt/ps3/petitpas/src/communication/box.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/synchro/sema.type"
#include "/mnt/ps3/petitpas/src/communication/box.type"

procedure putmess(box:string;var s:path;var mess:bufbloc;var succes:integer
{ Cette procedure ecrit un message dans la premiere place libre }

var
        de box:filbox;
        fd fdsem :fdok;
        ver : verrou;
        o1:ok;

{fonctions utilisees}
#include "/mnt/ps3/petitpas/src/synchro/sema.funct"

function open(s:string;m:openmode):fdok;extern;

function close(fd:fdok):ok;extern;

procedure getfbox(fd:fdok;var f:filbox);extern;

procedure putfbox(fd:fdok;var f:filbox);extern;

function wmess(fd:fdok;var f:filbox;var mess:bufbloc):ok;extern;

{ debut de PUTMESS }
begin
        { ouverture de la boite }
        fd:=open(box,rw);
        if (fd = -1)
        then
                succes:=1
        else
        begin
                { verrouillage }
                fdsem:=semopen(s);
                if (fdsem <> -1)
                then
                        ver:=lock(fdsem);
```

```
                        {lecture du descripteur }
                        getfbox( fd,desbox);
                        {existe-t-il des messages ?}
                        if (desbox.nbre = desbox.maxmes)
                        then
                                { plus de place }
                                succes:=3
                        else
                        begin
                                { fin de la boite ? }
                                if (desbox.ptwrite > desbox.maxmes)
                                then
                                        { on revient au debut }
                                        desbox.ptwrite:=1;


                                {ecriture du message }
                                o1:=wmess( fd,desbox,mess);
                                if (o1 = -1)
                                then
                                        { echec en ecriture }
                                        succes:=2
                                else
                                begin
                                        { mise-a-jour des pointeurs }
                                        desbox.ptwrite :=desbox.ptwrite+1;
                                        desbox.nbre:=desbox.nbre+1;
                                        {recopie du descripteur }
                                        putfbox( fd,desbox);
                                        succes:=0;
                                end;
                        end;
                        {deverrouillage}
                        if ( fdsem <> -1)
                        then
                                ver:=unlock( fdsem);
                        o1:=close( fdsem);
                        {fermeture }
                        o1:=close( fd);
                end;
        end;
```

Rmess.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
#include "/mnt/ps3/petitpas/src/communication/box.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/communication/box.type"
function rmess( fd:fdok;var desbox:filbox;var mess:bufbloc):ok;
{ Cette fonctiion lit un message a la place indique }

var
        pos,posb:integer;
        l:integer;
        ol:ok;

{fonctions utilisees }
function seek( fd:fdok;offset:integer;s:seekreq):ok;extern;

function uread( fd:fdok;var b:bufbloc;l:integer):integer;extern;

{debut de RMESS }
begin
        {positionnement}
        pos := desbox.lmes * (desbox.ptread -1 ) + ldesb;
        posb:= pos div lbloc;
        pos := pos -( lbloc * posb);
        ol:=seek( fd,posb,absb);
        ol:=seek( fd,pos,rel);
        if (ol = -1)
        then
                rmess:=-1
        else
        begin
                {lecture du message }
                l:=uread( fd,mess,desbox.lmes);
                if (l = -1)
                then
                        rmess:=-1
                else
                        rmess:=0;
        end;
end;
```

Wmess.p

```
#
{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
#include "/mnt/ps3/petitpas/src/communication/box.const"

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/communication/box.type"
function wmess(fd:fdok;var desbox:filbox;var mess:bufbloc):ok;
{ Cette fonction ecrit un message a la place indique }

var
        pos,posb:integer;
        l:integer;
        ol:ok;

{fonctions utilisees }
function seek(fd:fdok;offset:integer;s:seekreq):ok;extern;

function uwrite(fd:fdok;var b:bufbloc;l:integer):integer;extern;

{debut de WMESS }
begin
        {positionnement}
        pos := desbox.lmes * (desbox.ptwrite -1 ) + ldesb;
        posb:= pos div lbloc;
        pos := pos -(lbloc * posb);
        ol:=seek(fd,posb,absb);
        ol:=seek(fd,pos,rel);
        if (ol = -1)
        then
                wmess:=-1
        else
        begin
                {ecriture du message }
                l:=uwrite(fd,mess,desbox.lmes);
                if (l = -1)
                then
                        wmess:=-1
                else
                        wmess:=0;
        end;
end;
```

PRIMITIVES DE SYNCHRONISATION.

Types.

```
{ Déclaration des types propres aux sémaphores. }
verrou = (bloque, non_bloque);
path = string;
fd = 0 .. 14;
bsize = 1 .. 512;
bsizeok = -1 .. 512;
```

Dormez.p

```
{ Procédure d'endormissement momentané d'un processus }
type
        posint = 0 .. maxint;
procedure dormez (seconds : posint);
{ Primitive système de mise en état de repos d'un processus }
procedure sleep (seconds : posint); extern;
begin
sleep(seconds);
end;
```

Executez.p

```
#
{$c+}
{ Procédure de lancement d'exécution asynchrone d'un programme
  indépendant à partir d'un processus }
type
        path = string;
        arglist = array[1 .. 13] of string;
                { Last entry must be nil }
        pidok = -1 .. 32767;
        ok = -1 .. 0;
        fd = 0 .. 14;
procedure executez (prog : path);
var
        i : fd;
        close_ok : ok;
        arg : arglist;
{ Fonction système Unix de dédoublement de processus }
function fork : pidok; extern;
{ Procédure système Unix de substitution de processus }
procedure execv (p : path; var a : arglist); extern;
{ Fonction système Unix de fermeture  de fichier }
function close (f : fd) : ok; extern;
begin
if (fork = 0)
then begin
     { Nettoyage de la table, relais d'adressage des fichiers,
       associée au processus à créer; ce nettoyage est nécessaire
       du fait que le nouveau processus sera un processus
       enfant, donc qu'il héritera des fichiers ouverts par
       le processus générateur }
     for i := 3 to 14 do close_ok := close(i);
     { Exécution asynchrone du programme }
     arg[1] := nil;
     execv(prog, arg);
     end;
end;
```

Givecpt.p

```
#
#define fstasys "/mnt/ps3/petitpas/database/data/fstasys"
#include "/mnt/ps3/petitpas/src/synchro/sema.def"

{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
        lartproc = 18;

type
#include "/mnt/ps3/petitpas/src/access/acces.type"
        tnomf = packed array[1..16] of char;
        { record de fstasys }
        entree = record
                { nom du processus }
                nom : tnomf;
                { compteur du nombre de ce processus actif }
                cpt : integer;
        end;

        verrou = (bloque,non_bloque);
        path = string;

        fd = 0..14;

procedure givecpt(var nom : tnomf;var cpt : integer);
{ Cette procedure donne le compteur correspondant au nom de processus donne
   nom,il trouve dans le fichier fstasys ces informations }


var
        { nombre d'entree dans fstasys }
        nbproc : integer;
        artproc : entree;
        nomf : string;
        fds,fd : fdok;
        OK:ok;
        l : integer;
        trouve : boolean;
        sema : path;
        ver : verrou;

{ liste de sfonctions utilisees }
#include "/mnt/ps3/petitpas/src/synchro/sema.funct"

function uread(fd : fdok;var b:integer;l :posint):integer;extern;

function open(n:string;r : openmode):fdok;extern;
```

```
function close( fd:fdok ):ok;extern;

procedure getart( var artproc: entree );

function uread( fd:fdok;var b:entree;l:integer ):integer;extern;

begin
        l:=uread( fd,artproc,lartproc );
end;

{ debut de givecpt }
begin
        { ouverture de fstasys }
        nomf:=fstasys;
        fd:=open( nomf,rw );
        {verrouillage }
        sema:=s_fstasys;
        fds:=semopen( sema );
        ver:=lock( fds );

        { lecture du nombre d'entree dans le fichier }
        l:=uread( fd,nbproc,2 );

        { parcours des entrees }
        while ( nbproc > 0 )
        do
        begin
                { lecture d'un record }
                getart( artproc );
                { trouve ? }
                if ( artproc.nom = nom )
                then
                begin
                        trouve:=true;
                        nbproc:=0;
                        cpt:=artproc.cpt;
                end
                else
                        nbproc:=nbproc-1;
        end;

        { deverrouillage }
        ver:=unlock( fds );

        if not trouve
        then
                cpt:=-1;
        OK:=close( fds );
        OK:=close( fd );
end;
```

Lock.p

```
#
{$c+}
type
#include "/mnt/ps3/petitpas/src/synchro/sema.type"
{ Appel "système" utilisé pour manipuler un sémaphore;}
{  le deuxième paramètre de cette fonction est en fait}
{  l'adresse d'une zone tampon dans laquelle le résultat de}
{  la lecture doit être rangé. Mais, dans le cas précis des}
{  sémaphores, la fonction "uread" n'effectue pas de lecture;}
{  elle est en fait un astuce d'implémentation, c'est-à-dire}
{  que pour les adresses 0, 1, 2 et 3, la fonction "uread"}
{  réalise les fonctions "unlock", "lock", "lockif" et "locked".}
{  C'est pourquoi, le deuxième paramètre de "uread" est déclaré}
{  entier, afin de passer facilement les valeurs d'adresse }
{  0, 1, 2 et 3. }


{ Cette fonction verrouille un sém.....aphore, si celui-ci est}
{  non verrouillé, et attend pour le verrouiller jusqu'au moment du}
{  déverrouillage par un autre processus; de plus, elle renvoit}
{  l'état du sémaphore avant son intervention. }
function lock(s : fd) : verrou;

function uread(f : fd; b : integer; t : bsize) : bsizeok ; extern;
begin
if (uread(s,1,0) = 0)
then lock := non_bloque
else lock := bloque;
end;
```

Locked.P

```
#
{$c+}

type
#include "/mnt/ps3/petitpas/src/synchro/sema.type"

{ Appel "système" utilisé pour manipuler un sémaphore;
  le deuxième paramètre de cette fonction est en fait
  l'adresse d'une zone tampon dans laquelle le résultat de
  la lecture doit être rangé. Mais, dans le cas précis des
  sémaphores, la fonction "uread" n'effectue pas de lecture;
  elle est en fait un astuce d'implémentation, c'est-à-dire
  que pour les adresses 0, 1, 2 et 3, la fonction "uread"
  réalise les fonctions "unlock", "lock", "lockif" et "locked".
```

```
  C'est pourquoi, le deuxième paramètre de "uread" est déclaré
  entier, afin de passer facilement les valeurs d'adresse
  0, 1, 2 et 3. }

{ Cette fonction permet simplement de connaître l'état d'un
  sémaphore. }
function locked(s : fd) : verrou;

function uread(f : fd; b : integer; t : bsize) : bsizeok ; extern;
begin
if (uread(s,3,0) = 0)
then locked := non_bloque
else locked := bloque;
end;
```

Lockif.p

```
#
{$c+}

type
#include "/mnt/ps3/pe :itpas/src/synchro/sema.type"

{ Appel "système" utilisé pour manipuler un sémaphore;
  le deuxième paramètre de cette fonction est en fait
  l'adresse d'une zone tampon dans laquelle le résultat de
  la lecture doit être rangé. Mais, dans le cas précis des
  sémaphores, la fonction "uread" n'effectue pas de lecture;
  elle est en fait un astuce d'implémentation, c'est-à-dire
  que pour les adresses 0, 1, 2 et 3, la fonction "uread"
  réalise les fonctions "unlock", "lock", "lockif" et "locked".
  C'est pourquoi, le deuxième paramètre de "uread" est déclaré
  entier, afin de passer facilement les valeurs d'adresse
  0, 1, 2 et 3. }

{ Cette fonction verrouille un sémaphore sans attente et
  renvoit l'état de ce dernier avant intervention. }
function lockif(s : fd) : verrou;

function uread(f : fd; b : integer; t : bsize) : bsizeok ; extern;
begin
if (uread(s,2,0) = 0)
then lockif := non_bloque
else lockif := bloque;
end;
```

Majcpt.p

```
#
#define fstasys "/mnt/ps3/petitpas/database/data/fstasys"
#include "/mnt/ps3/petitpas/src/synchro/sema.def"

{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
        lartproc = 18;

type
#include  "/mnt/ps3/petitpas/src/access/acces.type"
        tnomf = packed array[1..16] of char;
        { record de fstasys }
        entree = record
                { nom du processus }
                nom : tnomf;
                { compteur du nombre de ce processus actif }
                cpt : integer;
        end;

        verrou = (bloque,non_bloque);
        path = string;

        fd = 0..14;

procedure majcpt(var nom:tnomf;var depl:integer);
{ Cette procure permet de mettre a jour le compteur d'un processus,
 cette mise a jour et relative (+1 -1) }


var
        { nombre d'entree dans fstasys }
        nbproc : integer;
        artproc : entree;
        nomf : string;
        fds,fd : fdok;
        OK:ok;
        l : integer;
        trouve : boolean;
        sema : path;
        ver : verrou;

{ liste des fonctions utilisees }
#include "/mnt/ps3/petitpas/src/synchro/sema.funct"

function uread(fd : fdok;var b:integer;l :posint):integer;extern;

function open(n:string;r : openmode):fdok;extern;
```

```
function close(fd:fdok):ok;extern;

function seek(fd:fdok;offset:integer;s:seekreq):ok;extern;

procedure getart(var artproc: entree);

function uread(fd:fdok;var b:entree;l:integer):integer;extern;

begin
        l:=uread(fd,artproc,lartproc);
end;

procedure putart(var artproc: entree);

function uwrite(fd:fdok;var b:entree;l:integer):integer;extern;

begin
        l:=uwrite(fd,artproc,lartproc);
end;

{ debut de majcpt }
begin
        { ouverture de fstasys }
        nomf:=fstasys;
        fd:=open(nomf,rw);
        {verrouillage }
        sema:=s_fstasys;
        fds:=semopen(sema);
        ver:=lock(fds);

        { lecture du nombre d'entree dans le fichier }
        l:=uread(fd,nbproc,2);

        { parcours des entrees }
        while (nbproc > 0)
        do
        begin
                { lecture d'un record }
                getart(artproc);
                { trouve ? }
                if (artproc.nom = nom )
                then
                begin
                        trouve:=true;
                        nbproc:=0;
                        artproc.cpt:=artproc.cpt+depl;
                        { reecriture }
                        OK:=seek(fd,-lartproc,rel);
                        putart(artproc);
                end
                else
                        nbproc:=nbproc-1;
        end;

        { deverrouillage }
```

```
        ver:=unlock( fds );


        OK:=close( fds );
        OK:=close( fd );
end;
```

Semopen.p

```
#

#define EN_LECTURE R

{$c+}
const
#include "/mnt/ps3/petitpas/src/access/acces.const"
type
#include "/mnt/ps3/petitpas/src/access/acces.type"
#include "/mnt/ps3/petitpas/src/synchro/sema.type"

{ Appel "système" utilisé pour l'ouverture des sémaphores
  ( ce ne sont pas des fichiers ordinaires ) }
function open(p : path; m : openmode) : fdok; extern;

{ Fonction d'ouvertur  d'un sémaphore implémenté dans UNIX }
function semopen(s :  ath) : fdok;

begin
semopen := open(s, EN_LECTURE);
end;
```

PRIMITIVES DIVERSES.

Compost.p

```
function compost(var cle :integer;var pas :integer):integer;
{ Cette fonction attribue une nouvelle cle par compostage
 elle ajoute a cle le pas.
  Cette fonction a ete specialement ecrite pour attribuer un numero de clie
  par compostage,a partir de la derniere cle du fichier,cle qui se trouve
  dans le descripteur de ce fichier.
}

begin
        cle :=cle + pas;
        compost:=cle;
end;
```

Clrlines.p

```
#
{$c+}

type

#include "/mnt/ps3/petitpas/src/application/video.type"

procedure clrlines(lfirst:integer;llast:integer);
{ Cette procedure efface un ensemble de lignes (de lfirst a llast)
  de l'ecran }

var
        i:integer;
{liste des procedures }
procedure posit(line,col:integer);extern;

procedure erase(zone:erasetype);extern;

{ debut de CLRLINES }
begin
        for i:=lfirst to llast
        do
        begin
                posit(i,1);
                erase(line);
        end;
end;
```

Readint.p

```
procedure readint (var ttyin : text; var n : integer;
                   var succes : boolean);

const
        lentier = 7;
        lligne = 80;

type
        lstring = 1 .. lligne;
        ligne = packed array[1 .. lligne] of char;
        posint = 1 .. maxint;

var
        i : integer;
        signe : integer;
        ordre : integer;
        l : ligne;

{ Procedure qui permet la lecture d'une chaîne de caractères
  au terminal }
procedure readstring (var tty : text; var s : ligne;
                      t : lstring); extern;

{ Procédure de calcul de la valeur du nombre }
procedure calcul;
begin
{ Sauter tous les espaces éventuels }
while (l[i] = ' ') and (i <= lentier)
do i := i + 1;
while (l[i] <> ' ') and (i <= lentier)
do begin
   ordre := ord(l[i]) - ord('0');
   if (ordre >= 0) and (ordre <= 9)
   then begin
        if (n <= (maxint - ordre) div 10)
        then n := n * 10 + ordre
        else begin
             succes := false;

             end;
        end
   else begin
        succes := false;

        end;
   i := i + 1;
   end;
end;
```

Readint.p

```
procedure readint (var ttyin : text; var n : integer;
                   var succes : boolean);

const
        lentier = 7;
        lligne = 80;

type
        lstring = 1 .. lligne;
        ligne = packed array[1 .. lligne] of char;
        posint = 1 .. maxint;

var
        i : integer;
        signe : integer;
        ordre : integer;
        l : ligne;

{ Procedure qui permet la lecture d'une chaîne de caractères
  au terminal }
procedure readstring (var tty : text; var s : ligne;
                      t : lstring); extern;

{ Procédure de calcul de la valeur du nombre }
procedure calcul;
begin
{ Sauter tous les espaces éventuels }
while (l[i] = ' ') and (i <= lentier)
do i := i + 1;
while (l[i] <> ' ') and (i <= lentier)
do begin
   ordre := ord(l[i]) - ord('0');
   if (ordre >= 0) and (ordre <= 9)
   then begin
        if (n <= (maxint - ordre) div 10)
        then n := n * 10 + ordre
        else begin
             succes := false;

             end;
        end
   else begin
        succes := false;

        end;
   i := i + 1;
   end;
end;
```

```
begin
readstring(ttyin, l, lentier);
succes := true;
n := 0;
signe := 1;
i := 1;
{ Sauter tous les espaces éventuels }
while (l[i] = ' ') and (i <= lentier)
do i := i + 1;
if (i > lentier)
then begin
     succes := false;
     end
else begin
     if (l[i] = '-')
     then begin
          signe := -1;
          i := i + 1;
          calcul;
          end
     else begin
          if (l[i] = '+')
          then begin
               i := i + 1;
               calcul;
               end
          else begin
               calcul;
               end;
          end;
     end;
n := n * signe;
end;
```

Readpint.p

```
type
        posint = 0 .. maxint;

procedure readpint (var ttyin : text; var n : posint;
                    var succes : boolean);

var
        entier : integer;

{ Procédure de lecture d'un entier au terminal }
procedure readint (var ttyin : text; var n : integer;
                    var succes : boolean); extern;

begin
readint(ttyin, entier, succes);
if succes
then begin
     if (entier >= 0)
     then n := entier
     else succes := false;
     end;
end;
```

Readstring.p

```
const

        lligne = 80;

type

        lstring = 1 .. lligne;
        ligne = packed array [1 .. lligne] of char;

procedure readstring (var tty : text; var l : ligne; t : lstring);

var

        i : integer;

begin
readln(tty);
for i := 1 to t
do if (eoln(tty))
   then for i := i to t
        do l[i] := ' '
   else read(tty,l[i]);
end;
```

ANNEXE 3 : MANUELS D'UTILISATION.

08/17/81

ACCESSES AND MANIPULATIONS OF A FILE.

DESCRIPTION

The 13 primitives describe in the following pages allows the direct access to a file with a identifying or non identifying key,a numerical or an alphanumerical key; and the sequential access to a file.

This system doesn't contain the synchronisation of the access between many processus and thus the protection of the file.

The file will contain fixed length records.A key is defined by her relative position in the record and her length in bytes in the record.If the position and the length are equal to zero it indicates the sequentiel access.

An access generates a file,the form of this file is the following : "name of your data file.XX",where XX is a number between 01 and 10.
Example : customer => customer.01,customer.02.

RESTRICTIONS

The maximum length of a record is 510 bytes.
The maximum length of a key is 508 bytes.
The maximum number of access is 10.

REMARKS

When you use a record which contains a packed array,you must use an even length for this array.

Filind.lastkey must be filled by the programmer.

TYPES AND CONSTANTS

```
const
        { maximum length of a record }
        lmaxrec = 510;
        { maximum number of access for a file }
        maxacc = 10;
        { maximum length for a key }
        lmaxkey = 508;
```

```
        { maximum number of record in a file }
        maxrec = 32765;


type
        { type of key }
        typezone = (C,E);

        { type for the record }
        data = packed array [1 .. lmaxrec] of char;

        { type for the key }
        key = packed array [1..lmaxkey] of char;

        posint = 0..maxint;

        nposint = -1..maxint;

        { file descriptor for UNIX }
        fdok = -1..14;


        { description of a key }
        keyacces = record
                { position of the key in the record }
                poskey : nposint;
                { length of the key }
                lengkey : nposint;
                { type of key }
                type_key : typezone;
        end;

        descript = record
                { length of the record in bytes }
                lrec : posint;
                { number of records in the file }
                nbrec : posint;
                { number of deleted records in the file }
                recdel : posint;
                { length of this descriptor in bytes }
                ldesfile:integer;
                { list of the key }
                lsacces : array [1..maxacc] of cleacces;
        end;

        { descriptor of the file }
        filcont = record
                { name of the file }
                name : string;
                { file descriptor for UNIX }
                fd : fdok;
                { header }
                header:descript;
        end;
```

```
{ descriptor of an access  }
filind = record
        { position of the key in the record }
        poskey : nposint;
        { length of the key in bytes }
        lengkey : nposint;
        { identifying access ? }
        ac_ident:boolean;
        { type of key }
        type_key:typezone;
        { file descriptor for UNIX }
        fd : fdok;
        { length of the access record }.
        lrec : posint;
        { number of access records }
        nbrec : posint;
        { length of this descriptor in bytes }
        ldesind:integer;
         { last key }
        lastkey : key;
end;
```

FILE

The library is : /mnt/ps3/petitpas/bin/laccess for the compiled version.

The Pascal source version is in /mnt/ps3/petitpas/src/access.

NAME

closeac - close an access

SYNOPSIS

```
procedure closeac(var desdon : filcont;var desind :  filind;
var result : integer);
```

DESCRIPTION

Closeac closes an access to a file.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :
1 if the close has failed
2 if the access isn't created
3 if the file isn't opened.

SEE ALSO

Closef, compact, consult, createac, createf, delete, deleteac, insert, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

closef - close a file

SYNOPSIS

```
procedure closef(var desdon : filcont;var result : integer);
```

DESCRIPTION

Closef closes a file.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :
1 if the close has failed
2 if the file isn't opened.

SEE ALSO

Closeac, compact, consult, createac, createf, delete, deleteac, insert, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

compact - compact a file

SYNOPSIS

```
procedure compact(var name : string;var rate  :  integer;var
result : integer; var output : text);
```

DESCRIPTION

Compact compacts a file.In this system,the deleted records aren't physically deleted from the file,thus we offer a procedure which can compact the file "name".The "rate" (%) is the percentage of deleted records from which the file must be compacted.

The file "name" and all his accesses must be closed before the operation.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :

1 if it isn't necessary to compact the file

2 if a problem occurs during the manipulation of the file

3 if a problem occurs during the manipulation of an access.

SEE ALSO

Closeac, closef, consult, createac, createf, delete, deleteac, insert, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

consult - consult a file

SYNOPSIS

```
procedure consult(var desdon : filcont;var desind :
filind;var key : key; var record : data;var result :
boolean);
```

DESCRIPTION

Consult gives the first record matching to the given key if such a record exists.

DIAGNOSTICS

A result true indicates that no problem occurs.

If the operation has failed,the result is false.

The problems can be :
- the file isn't opened
- the access isn't opened
- the key doesn't exist
- the record is deleted.

SEE ALSO

Closeac, closef, compact, createac, createf, delete, deleteac, insert, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

createac - creat an access to a file

SYNOPSIS

```
procedure createac(var desdon : filcont;var desind : filind;
var result : integer);
```

DESCRIPTION

Createac creats and opens an access to a file. The user must fill "desind.poskey" with the relative position of the key in the record;"desind.lengkey" with the length in bytes of the key; "desind.ac_ident" : true for a identifying key and false for an non identifying key; "desind.type_key" : E indicates an integer key and C indicates an alphanumerical key.

For "desind.poskey",we must have :
0 <= "desind.poskey" <= "desind.header.lrec",

for "desind.lengkey",we must have :
0 <= "desind.lengkey" <= lmaxkey(508).

If "desind.poskey" and "desind.lengkey" are equal to zero,it indicates a sequentiel access to the file. For this access,"desind.ac_ident" must be false.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :
1 if the file isn't opened
2 if the length of the key is too long
3 if the position of the key is incorrect
4 if the key isn't in the record,we must have :
"desind.poskey" + "desind.lengkey" <=
"desdon.header.lrec"
5 if it is impossible to create another access (maximum maxacc(10) accesses)
6 if the creation has failed
7 if the access exists already.

SEE ALSO

Closeac, closef, compact, consult, createf, delete, deleteac, insert, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

createf - create a file

SYNOPSIS

```
procedure createf(var desdon : filcont;var result :
integer);
```

DESCRIPTION

Createf creates and opens a file. The user must fill "desdon.name" with the name of the file, and "desdon.header.lrec" with the length in bytes of the record. The length is fixed and must respect the following inequations :
0 < "desdon.header.lrec" <= lmaxrec (510).

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :
1 if the length is incorrect
2 if the creation has failed.

SEE ALSO

Closeac, closef, compact, consult, createac, delete, deleteac, insert, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

delete - delete a record from a file

SYNOPSIS

```
procedure delete(var desdon : filcont;var desind :
filind;var key : key; var result : boolean);
```

DESCRIPTION

Delete deletes the record matching to the given key if such a record exists; but the key must be an identifying one.

All the other accesses to the file must be closed.

DIAGNOSTICS

A result true indicates that no problem occurs.

If the operation has failed,the result is false.

The problems can be :
- the file isn't opened
- the access isn't opened
- the access isn't an identifying one
- the key doesn't exist
- a problem occurs during the manipulation of an another access.

SEE ALSO

Closeac, closef, compact, consult, createac, createf, deleteac, insert, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

deleteac - delete an access

SYNOPSIS

```
procedure deleteac(var desdon : filcont;var desind : filind;
var result : integer);
```

DESCRIPTION

Deleteac deletes an access to a file.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is 1.

SEE ALSO

Closeac, closef, compact, consult, createac, createf, delete, insert, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

insert - insert a record in a file

SYNOPSIS

```
procedure insert(var desdon : filcont;var record :  data;var
result : boolean);
```

DESCRIPTION

Insert inserts the given record into the file and updates all access associated to the file. The user must fill the variable "record".The variable "desdon" contents the informations filled by a preceding call to the procedure "openf" or "createf".

All the accesses to the file must be closed.

DIAGNOSTICS

A result true indicates that no problem occurs.

If the operation has failed,the result is false.

The problems can be :
- the file isn't opened
- the file is full
- a problem occurs during the manipulation of an access.

SEE ALSO

Closeac, closef, compact, consult, createac, createf, delete, deleteac, modify, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

modify - modify a record of a file

SYNOPSIS

```
procedure modify(var desdon : filcont;var desind :
filind;var record : data; var result : boolean);
```

DESCRIPTION

Modify modifies the current given record of the file.

DIAGNOSTICS

A result true indicates that no problem occurs.

If the operation has failed,the result is false.

The problems can be :
- the file isn't opened
- the access isn't opened
- a key,in the record,has been modified.

SEE ALSO

Closeac, closef, compact, consult, createac, createf, delete, deleteac, insert, openac, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

openac - open an access

SYNOPSIS

```
procedure openac(var desdon : filcont;var desind  :  filind;
var result : integer);
```

DESCRIPTION

Openac opens an access to a file. The user must fill "desind.poskey" with the relative position of the key in the record and "desind.lengkey" with the length in bytes of the key.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :
1 if the open has failed
2 if the access isn't created
3 if the file isn't opened.

SEE ALSO

Closeac, closef, compact, consult, createac, createf, delete, deleteac, insert, modify, openf, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

openf - open a file

SYNOPSIS

```
procedure openf(var desdon : filcont;var result : integer);
```

DESCRIPTION

Openf opens a file. The user must fill "desdon.name" wih the name of the file.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is 1.

SEE ALSO

Closeac, closef, compact, consult, createac, createf, delete, deleteac, insert, modify, openac, readnext.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

readnext - read the next record of a file

SYNOPSIS

```
procedure readnext(var desdon : filcont;var desind :
filind;var record : data; var eof : boolean;var result :
boolean);
```

DESCRIPTION

Readnext reads the following record of the file,if such a record exists, according to the given key and the current record.The given key may be the sequential classical access or an other one.

DIAGNOSTICS

A result true indicates that no problem occurs.

If the operation has failed,the result is false.

The problems can be :
- the file isn't opened
- the access isn't opened.

When the result is true,the indicator "eof" has always the false value. When the result is false,if the raison of the failing is the reaching of the end of file then the indicator "eof" has the value true else false.

SEE ALSO

Closeac, closef, compact, consult, createac, createf, delete, deleteac, insert, modify, openac, openf.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

08/17/81

COMMUNICATION OF MESSAGES BETWEEN PROCESSUS.

DESCRIPTION

The four primitives describe on the following pages allows the communication of messages between an illimited number of processus.But with this system, a processus can't warn another processus of the existence of a message. The processus can get or put fixed length messages in a box,which is a file.

RESTRICTIONS

The maximum length of the messages is 512 bytes.

REMARK

When you use a record which contains a packed array,you must use an even length for this array.

TYPES AND CONSTANTS

```
const
        { length of a message }
        lmes = 512;


type
        { type for the messages }
        bufblock = packed array [1..lmes] of char;

        { path }
        path = string;
```

FILES

The libraries are : /mnt/ps3/petitpas/bin/lcom and /mnt/ps3/petitpas/bin/lsynchro for the compiled version.

The Pascal source version is in /mnt/ps3/petitpas/src/communication and /mnt/ps3/petitpas/src/synchro.

NAME

creatbox - creat a box for the communication of messages between processus

SYNOPSIS

```
function creatbox(box_name : string;length_mess : integer):
integer;
```

DESCRIPTION

This function creats the box "box_name" for the communication of messages between processus.

This box will contain messages with the fixed lenght "lenght_mess" in bytes ,this lenght must respect the inequations :

```
0 < "lenght_mess" <= lmes(512)
```

If the box exists already,the function doesn't create it again.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :

1 if the box exists already
2 if the lenght is incorrect
3 if the creation has failed.

SEE ALSO

Delbox,getmess,putmess.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

delbox - delete a box

SYNOPSIS

```
function delbox(box_name : string):integer;
```

DESCRIPTION

This function deletes a box even if there is still messages in the box.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is 1.

SEE ALSO

Creatbox,getmess,putmess.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

getmess - get a message from a box

SYNOPSIS

```
procedure getmess(box_name : string;var  lock  :  path;  var
mess : bufblock;var result : integer);
```

DESCRIPTION

This procedure gets a message "mess" from a box "box_name" and removes it.The user must associate a lock ("lock") with this box.

The management of the messages is FIFO.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :
- 1 if the box doesn't exist
- 2 if a physical problem occurs
- 3 if there are no more messages in the box.

SEE ALSO

Creatbox,delbox,putmess.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

NAME

putmess - put a message in a box

SYNOPSIS

```
procedure putmess(box_name : string;var lock : path; var mes
: bufblock;var result : integer);
```

DESCRIPTION

This procedure puts a message "mess" in the box "box_name".The user must associate a lock ("lock") with this box.

The management of the messages is FIFO.

DIAGNOSTICS

A result 0 indicates that no problem occurs.

If the operation has failed,the result is :
- 1 if the box doesn't exist
- 2 if a physical problem occurs
- 3 if there is no more places in the box to put this message.

SEE ALSO

Creatbox,delbox,getmess.

AUTHORS

Bodart Jean-Pierre and Tixhon Jean-Paul (FUNDP Namur).

Système d'Informations de Petitpas.

Manuel du responsable du système.

Démarrage du système.

L'ordinateur doit être mis en route pour que l'application Petitpas puisse se dérouler.

Arrêt du système.

1.Entrer dans /mnt/ps3/petitpas/bin/application.

2. Attendre que tous les programmes de l'application soient terminés : lancer la commande TESTSTOP pour connaître l'état des programmmes.

3. Effectuer un sauvetage des fichiers : lancer la commande BACKUP.

4. Effectuer le compactage des fichiers : lancer la commande MAINTFIC et répondre aux questions posées.

5. Si le compactage s'est déroulé sans problèmes effectuer un sauvetage (BACKUP) sinon analyser les messages .

La commande RESTORE permet de restaurer la base de données.

Système d'Informations de Petitpas.

Manuel utilisateurs.

Accès au système.

(à partir d'un terminal vt100 seulement)

l'utilisateur	le système
	log in :
1.taper : petitpas<return>	
	password :
2.taper : le mot de passe <return>	

L'utilisateur se trouve alors dans le Système d'Informations Petitpas, il lui suffit de répondre aux questions posées.

Sortie du système.

Le système se charge lui-même de cloturer la session dès que l'utilisateur a terminé celle-ci.